

IF4073 Pemrosesan Citra Digital

DOKUMEN TUGAS KECIL 3: SEGMENTASI OBJEK DAN DETEKSI JALUR GARIS



Oleh:

13521042 Kevin John Wesley Hutabarat

13521140 Ryan Samuel Chandra

**PROGRAM STUDI TEKNIK INFORMATIKA
SEKOLAH TEKNIK ELEKTRO DAN INFORMATIKA
INSTITUT TEKNOLOGI BANDUNG
2024**

DAFTAR ISI

DAFTAR ISI.....	2
DAFTAR GAMBAR.....	3
I. Hasil Tangkap Layar GUI.....	6
II. Segmentasi Objek.....	6
III. Deteksi Jalur Garis.....	40
IV. Alamat GitHub Program.....	52

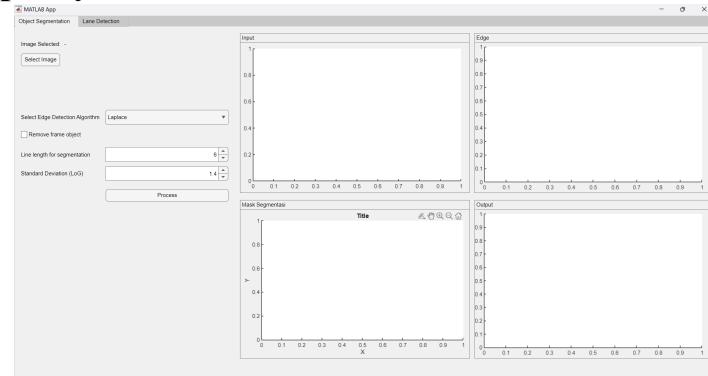
DAFTAR GAMBAR

Gambar 1.1 Tampilan GUI untuk fitur segmentasi objek.....	6
Gambar 1.2 Tampilan GUI untuk fitur deteksi jalur.....	6
Gambar 2.1 Segmentasi objek dari citra koin.jpg menggunakan Operator Laplace.....	9
Gambar 2.2 Segmentasi objek dari citra persik.jpg menggunakan Operator Laplace.....	9
Gambar 2.3 Segmentasi objek dari citra mobil.jpg menggunakan Operator Laplace.....	10
Gambar 2.4 Segmentasi objek dari citra rumah.jpg menggunakan Operator Laplace.....	10
Gambar 2.5 Segmentasi objek dari citra alpukat.jpg menggunakan Operator Laplace.....	10
Gambar 2.6 Segmentasi objek dari citra plat-kendaraan.jpg menggunakan Operator Laplace.....	11
Gambar 2.7 Segmentasi objek dari citra pisang.jpg menggunakan Operator Laplace.....	11
Gambar 2.8 Segmentasi objek dari citra mobil-motor.jpg menggunakan Operator Laplace.....	11
Gambar 2.9 Segmentasi objek dari citra kursi.jpg menggunakan Operator Laplace.....	12
Gambar 2.10 Segmentasi objek dari citra butterfly.jpg menggunakan Operator Laplace.....	12
Gambar 2.11 Segmentasi objek dari citra lotus.jpg menggunakan Operator Laplace.....	12
Gambar 2.12 Segmentasi objek dari citra portrait.jpg menggunakan Operator Laplace.....	13
Gambar 2.13 Segmentasi objek dari citra koin.jpg menggunakan Operator LoG.....	14
Gambar 2.14 Segmentasi objek dari citra persik.jpg menggunakan Operator LoG.....	15
Gambar 2.15 Segmentasi objek dari citra mobil.jpg menggunakan Operator LoG.....	15
Gambar 2.16 Segmentasi objek dari citra rumah.jpg menggunakan Operator LoG.....	15
Gambar 2.17 Segmentasi objek dari citra alpukat.jpg menggunakan Operator LoG.....	15
Gambar 2.18 Segmentasi objek dari citra mobil.jpg menggunakan Operator LoG.....	16
Gambar 2.19 Segmentasi objek dari citra pisang.jpg menggunakan Operator LoG.....	16
Gambar 2.20 Segmentasi objek dari citra mobil-motor.jpg menggunakan Operator LoG.....	16
Gambar 2.21 Segmentasi objek dari citra kursi.jpg menggunakan Operator LoG.....	17
Gambar 2.22 Segmentasi objek dari citra butterfly.jpg menggunakan Operator LoG.....	17
Gambar 2.23 Segmentasi objek dari citra lotus.jpg menggunakan Operator LoG.....	17
Gambar 2.24 Segmentasi objek dari citra kursi.jpg menggunakan Operator LoG.....	18
Gambar 2.25 Segmentasi objek dari citra koin.jpg menggunakan Operator Sobel.....	20
Gambar 2.26 Segmentasi objek dari citra persik.jpg menggunakan Operator Sobel.....	20
Gambar 2.27 Segmentasi objek dari citra mobil.jpg menggunakan Operator Sobel.....	20
Gambar 2.28 Segmentasi objek dari citra rumah.jpg menggunakan Operator Sobel.....	21
Gambar 2.29 Segmentasi objek dari citra alpukat.jpg menggunakan Operator Sobel.....	21
Gambar 2.30 Segmentasi objek dari citra plat-kendaraan.jpg menggunakan Operator Sobel.....	21
Gambar 2.31 Segmentasi objek dari citra pisang.jpg menggunakan Operator Sobel.....	22

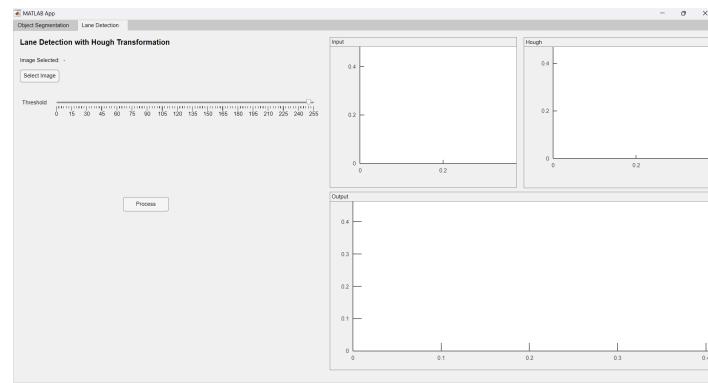
Gambar 2.32 Segmentasi objek dari citra mobil-motor.jpg menggunakan Operator Sobel.....	22
Gambar 2.33 Segmentasi objek dari citra kursi.jpg menggunakan Operator Sobel.....	22
Gambar 2.34 Segmentasi objek dari citra butterfly.jpg menggunakan Operator Sobel.....	23
Gambar 2.35 Segmentasi objek dari citra lotus.jpg menggunakan Operator Sobel.....	23
Gambar 2.36 Segmentasi objek dari citra portrait3.jpg menggunakan Operator Sobel.....	23
Gambar 2.37 Segmentasi objek dari citra koin.jpg menggunakan Operator Prewitt.....	25
Gambar 2.38 Segmentasi objek dari citra persik.jpg menggunakan Operator Prewitt.....	26
Gambar 2.39 Segmentasi objek dari citra mobil.jpg menggunakan Operator Prewitt.....	26
Gambar 2.40 Segmentasi objek dari citra rumah.jpg menggunakan Operator Prewitt.....	26
Gambar 2.41 Segmentasi objek dari citra alpukat.jpg menggunakan Operator Prewitt.....	27
Gambar 2.42 Segmentasi objek dari citra plat-kendaraan.jpg menggunakan Operator Prewitt.....	27
Gambar 2.43 Segmentasi objek dari citra pisang.jpg menggunakan Operator Prewitt.....	27
Gambar 2.44 Segmentasi objek dari citra mobil-motor.jpg menggunakan Operator Prewitt.....	28
Gambar 2.45 Segmentasi objek dari citra kursi.jpg menggunakan Operator Prewitt.....	28
Gambar 2.46 Segmentasi objek dari citra butterfly.jpg menggunakan Operator Prewitt.....	28
Gambar 2.47 Segmentasi objek dari citra lotus.jpg menggunakan Operator Prewitt.....	29
Gambar 2.36 Segmentasi objek dari citra portrai3.jpg menggunakan Operator Prewitt.....	29
Gambar 2.49 Segmentasi objek dari citra koin.jpg menggunakan Operator Roberts.....	31
Gambar 2.50 Segmentasi objek dari citra persik.jpg menggunakan Operator Roberts.....	31
Gambar 2.51 Segmentasi objek dari citra mobil.jpg menggunakan Operator Roberts.....	31
Gambar 2.52 Segmentasi objek dari citra rumah.jpg menggunakan Operator Roberts.....	32
Gambar 2.53 Segmentasi objek dari citra alpukat.jpg menggunakan Operator Roberts.....	32
Gambar 2.54 Segmentasi objek dari citra plat-kendaraan.jpg menggunakan Operator Roberts.....	32
Gambar 2.55 Segmentasi objek dari citra pisang.jpg menggunakan Operator Roberts.....	32
Gambar 2.56 Segmentasi objek dari citra mobil-motor.jpg menggunakan Operator Roberts.....	33
Gambar 2.57 Segmentasi objek dari citra kursi.jpg menggunakan Operator Roberts.....	33
Gambar 2.58 Segmentasi objek dari citra butterfly.jpg menggunakan Operator Roberts.....	33

Gambar 2.59 Segmentasi objek dari citra lotus.jpg menggunakan Operator Roberts.....	34
Gambar 2.60 Segmentasi objek dari citra portrait.jpg menggunakan Operator Roberts.....	34
Gambar 2.61 Segmentasi objek dari citra koin.jpg menggunakan Operator Canny.....	35
Gambar 2.62 Segmentasi objek dari citra persik.jpg menggunakan Operator Canny.....	36
Gambar 2.63 Segmentasi objek dari citra mobil.jpg menggunakan Operator Canny.....	36
Gambar 2.64 Segmentasi objek dari citra rumah.jpg menggunakan Operator Canny.....	36
Gambar 2.65 Segmentasi objek dari citra alpukat.jpg menggunakan Operator Canny.....	36
Gambar 2.66 Segmentasi objek dari citra plat-kendaraan.jpg menggunakan Operator Canny.....	37
Gambar 2.67 Segmentasi objek dari citra pisang.jpg menggunakan Operator Canny.....	37
Gambar 2.68 Segmentasi objek dari citra mobil-motor.jpg menggunakan Operator Canny.....	37
Gambar 2.69 Segmentasi objek dari citra kursi.jpg menggunakan Operator Canny.....	38
Gambar 2.70 Segmentasi objek dari citra butterfly.jpg menggunakan Operator Canny.....	38
Gambar 2.71 Segmentasi objek dari citra lotus.jpg menggunakan Operator Canny.....	38
Gambar 2.72 Segmentasi objek dari citra portrait.jpg menggunakan Operator Canny.....	39
Gambar 3.1 Deteksi Garis Jalur dari Citra img35.jpg dengan Bantuan Transformasi Hough (T=225).....	44
Gambar 3.2 Deteksi Garis Jalur dari Citra img39.jpg dengan Bantuan Transformasi Hough (T=240).....	45
Gambar 3.3 Deteksi Garis Jalur dari Citra img40.jpg dengan Bantuan Transformasi Hough (T=255).....	46
Gambar 3.4 Deteksi Garis Jalur dari Citra img57.jpg dengan Bantuan Transformasi Hough (T=255).....	47
Gambar 3.5 Deteksi Garis Jalur dari Citra img58.jpg dengan Bantuan Transformasi Hough (T≈70).....	48
Gambar 3.6 Deteksi Garis Jalur dari Citra road1.jpg dengan Bantuan Transformasi Hough (T=255).....	49
Gambar 3.7 Deteksi Garis Jalur dari Citra road2.jpg dengan Bantuan Transformasi Hough (T=255).....	50
Gambar 3.8 Deteksi Garis Jalur dari Citra road3.jpg dengan Bantuan Transformasi Hough (T=150).....	51

I. Hasil Tangkap Layar GUI



Gambar 1.1 Tampilan GUI untuk fitur segmentasi objek



Gambar 1.2 Tampilan GUI untuk fitur deteksi jalur

Gambar 1.1 dan Gambar 1.2 menunjukkan tampilan GUI masing-masing untuk fitur segmentasi objek dan deteksi jalur. Pada fitur segmentasi objek, GUI menampilkan citra input, citra tepi, citra mask objek, dan citra hasil deteksi objek. Fitur ini menerima parameter berupa status penghapusan objek pada tepi citra bertipe boolean dan panjang garis dilatasi untuk segmentasi objek bertipe *double*. Untuk deteksi tepi dengan Laplacian of Gaussian, program menerima parameter berupa simpangan baku, yang digunakan untuk membentuk mask konvolusi. Untuk fitur deteksi jalur garis, GUI menampilkan citra input, citra hasil deteksi jalur, dan citra transformasi Hough. Fitur ini menerima parameter berupa *threshold* yang akan digunakan untuk mendeteksi garis.

II. Segmentasi Objek

Segmentasi objek adalah suatu proses yang dilakukan untuk mendeteksi objek pada suatu citra melalui bentuknya. Pada pelaksanaan ini, segmentasi dilakukan dengan memisahkan objek dari latar belakangnya, kemudian menampilkan objek tersebut pada citra terpisah. Segmentasi objek yang dilakukan pada tugas ini memanfaatkan citra tepi dari sebuah citra. Untuk melakukan segmentasi objek dengan citra tepi, diperlukan input berupa citra asli dan citra hasil deteksi tepi. Citra tepi yang digunakan haruslah citra biner. Citra tepi terkadang tidak dalam keadaan yang tersambung. Oleh karena itu, citra

tepi harus diolah terlebih dahulu. Pengolahan ini dilakukan dengan proses yang bernama dilasi, yaitu mempertebal garis-garis tepi. Hal ini dilakukan agar garis-garis yang putus dapat tersambung dengan baik. Setelah garis tepi tersambung dengan baik, celah-celah kosong yang berada di dalam objek akan diisi. Setelah pemrosesan awal citra tepi ini selesai, citra tersebut akan di-*masking* ke citra asli. *Pixel-pixel* yang bersesuaian dengan citra tepi akan dipertahankan, sementara yang lainnya akan diabaikan.

Berikut adalah implementasi segmentasi objek yang dilakukan di MATLAB.

```
% segmentation.m
function [imgout, BWfiltered] = segmentation(imgin, imgoriginal,
clearBorder, lineLength)
    % referensi:
    https://www.mathworks.com/help/images/detecting-a-cell-using-image-segmentation.html
    lineLength = lineLength; % Panjang elemen struktural untuk dilasi garis
    minObjectSize = 4000; % Ukuran minimum area yang akan ditampilkan

    %Cek tipe data imgin
    if ~islogical(imgin)
        if size(imgin, 3) == 3
            imgin = rgb2gray(imgin); % Konversi ke grayscale jika input RGB
        end
        % Thresholding global untuk gambar biner
        imgin = imbinarize(imgin);
    end

    % Clear border untuk menghilangkan objek di tepi gambar
    if (clearBorder)
        dilated = imclearborder(imgin);
    else
        dilated = imgin;
    end
    % Penguatan garis (Dilasi dan Closing)
    se0 = strel('line', lineLength, 0); % Garis horizontal
    se90 = strel('line', lineLength, 90); % Garis vertikal
    BWdilated = imdilate(dilated, [se90 se0]); % Dilasi untuk menyambungkan garis
    % Closing untuk menutup celah di dalam objek
    BWclosed = imclose(BWdilated, strel('disk', 2));
    % Mengisi area kosong di dalam objek
    BWfilled = imfill(BWclosed, 'holes');
    % Penghapusan objek kecil yang tidak relevan
    BWfiltered = bwareaopen(BWfilled, minObjectSize);
    % Masking ke gambar asli
    imgout = bsxfun(@times, imgoriginal, cast(BWfiltered, 'like',
    imgoriginal));
end
```

Dapat dilihat bahwa fungsi segmentasi menerima parameter citra asli dan citra tepi. Segmentasi objek yang dilakukan pada tugas ini memanfaatkan pendekatan objek. Berikut adalah beberapa operasi pendekatan tepi yang digunakan pada tugas ini.

1. Operasi Laplace

Operasi Laplace adalah operasi yang dibuat dengan memanfaatkan pendekatan kalkulus diferensial. Hal ini didasarkan pada perubahan intensitas yang besar dalam jarak yang singkat dapat dipandang sebagai fungsi yang memiliki kemiringan yang besar. Operator Laplace memanfaatkan turunan kedua untuk mendekati lokasi tepi yang lebih akurat, khususnya pada tepi yang curam. Penurunan rumus dari turunan kedua ini dapat disederhanakan menjadi matriks, yang salah satunya adalah seperti matriks di bawah ini.

$$\begin{bmatrix} 1 & 1 & 1 \\ 1 & -8 & 1 \\ 1 & 1 & 1 \end{bmatrix}$$

Berikut adalah kode implementasi Operasi Laplace

```
% laplace.m
function output = laplace(image)
    % Kernel Laplace
    kernel = [1 1 1; 1 -8 1; 1 1 1];

    % konvolusi
    % fungsi ini didefinisikan sendiri pada file convolution.m
    output = convolution(double(image), double(kernel));

    % Normalize to 0-255
    output = uint8(mat2gray(output) * 255);
end

% convolution.m
function outputImage = convolution(image, convMatrix)
    [rows, cols, colorChannels] = size(image);
    [rowsConv, colsConv] = size(convMatrix);

    outputImage = zeros(rows, cols, 'uint8');
    for i = ((rowsConv+1)/2):(rows-((rowsConv-1)/2))
        for j = ((colsConv+1)/2):(cols-((colsConv-1)/2))
            for k = 1:colorChannels
                temp = 0;
                startIdxRow = i-(rowsConv+1)/2;
                startIdxCols = j-(colsConv+1)/2;
                for a = 1:rowsConv
                    for b = 1:colsConv
                        temp = temp + image(a + startIdxRow, b +
```

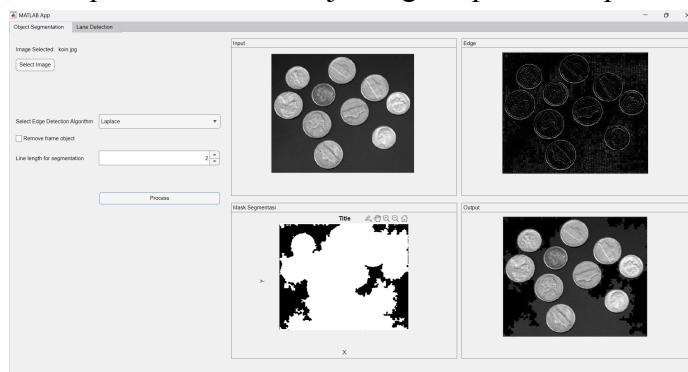
```

startIdxCols,k)*convMatrix(a,b);
    end
end

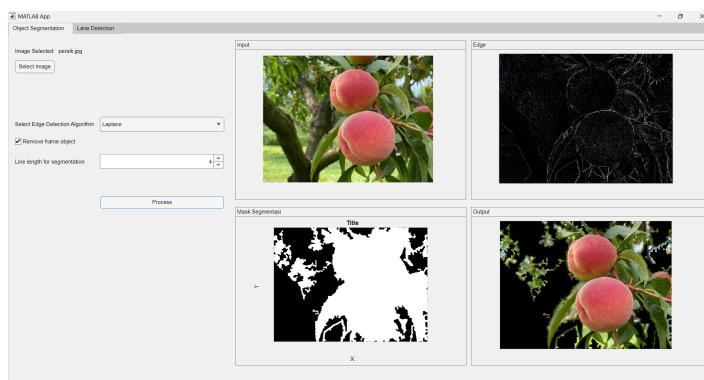
if temp < 0
    outputImage(i,j,k) = 0;
elseif temp > 255
    outputImage(i,j,k) = 255;
else
    outputImage(i,j,k) = uint8(temp);
end
end
end

```

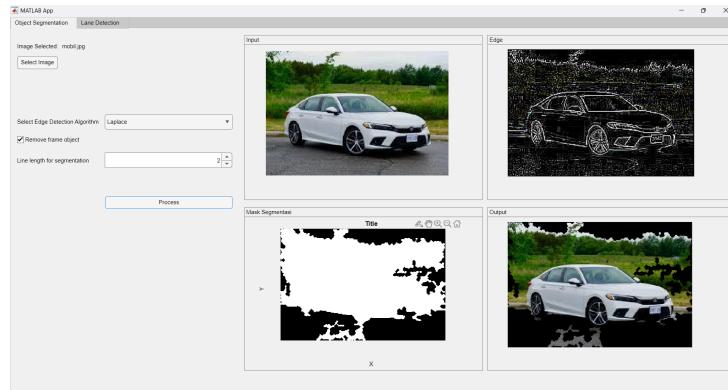
Berikut adalah hasil pemrosesan citra uji dengan operator Laplace.



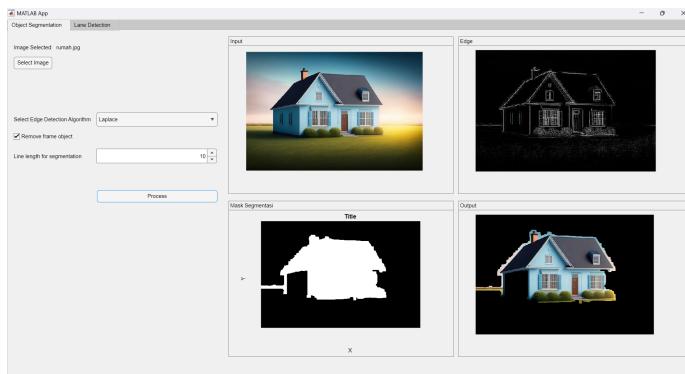
Gambar 2.1 Segmentasi objek dari citra koin.jpg menggunakan Operator Laplace



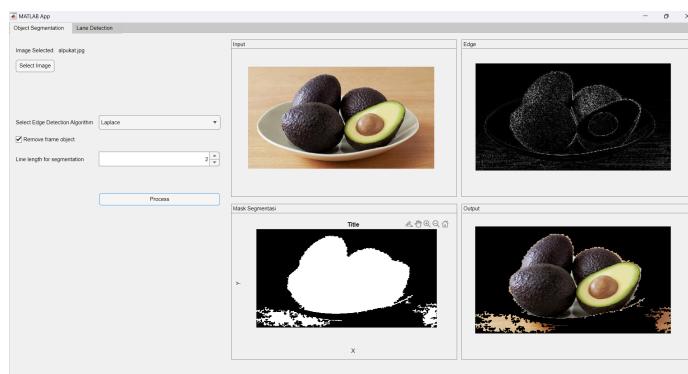
Gambar 2.2 Segmentasi objek dari citra persik.jpg menggunakan Operator Laplace



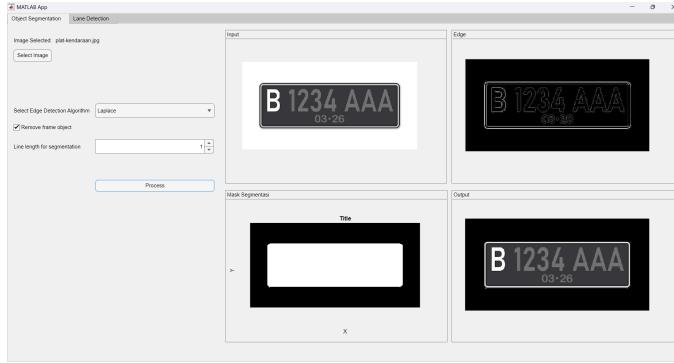
Gambar 2.3 Segmentasi objek dari citra mobil.jpg menggunakan Operator Laplace



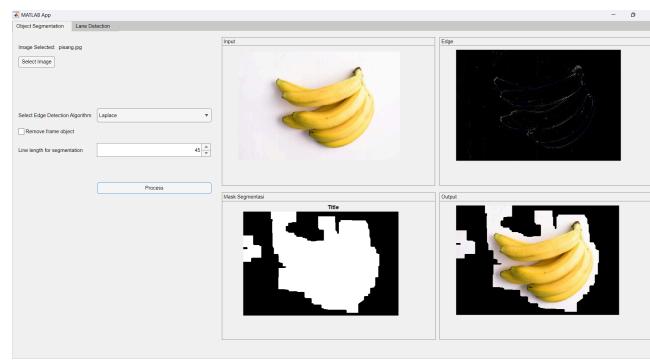
Gambar 2.4 Segmentasi objek dari citra rumah.jpg menggunakan Operator Laplace



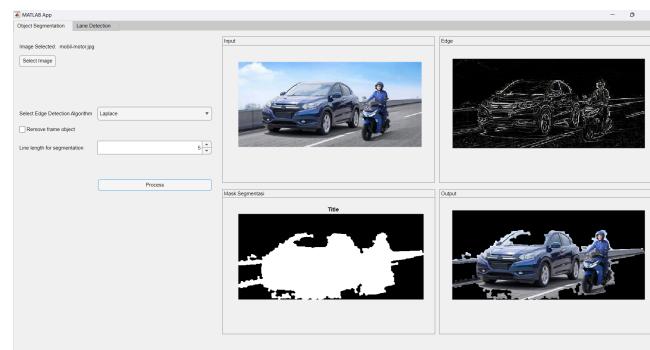
Gambar 2.5 Segmentasi objek dari citra alpukat.jpg menggunakan Operator Laplace



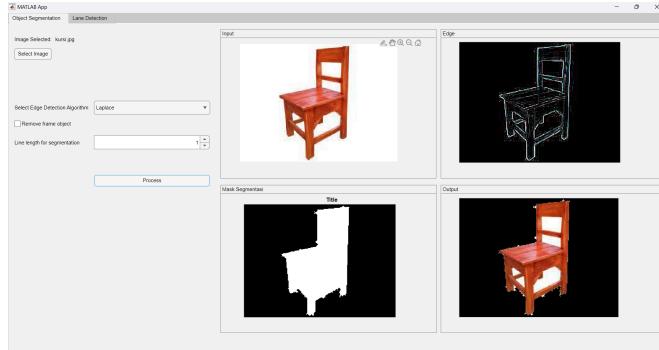
Gambar 2.6 Segmentasi objek dari citra plat-kendaraan.jpg menggunakan Operator Laplace



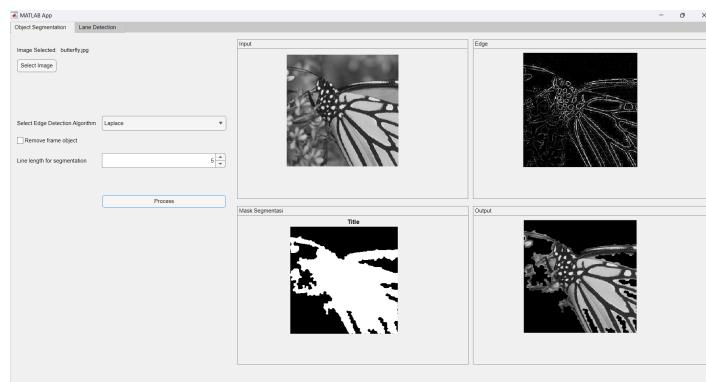
Gambar 2.7 Segmentasi objek dari citra pisang.jpg menggunakan Operator Laplace



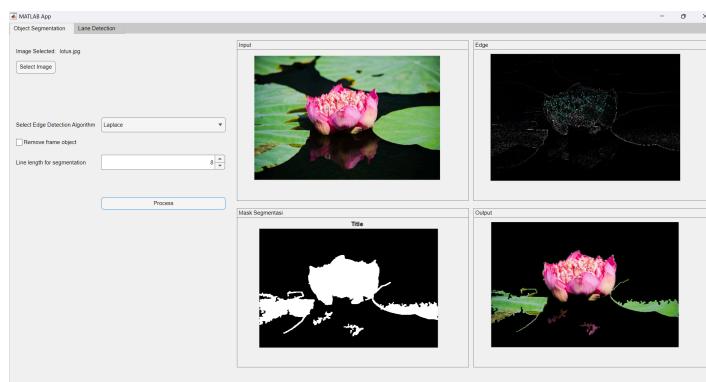
Gambar 2.8 Segmentasi objek dari citra mobil-motor.jpg menggunakan Operator Laplace



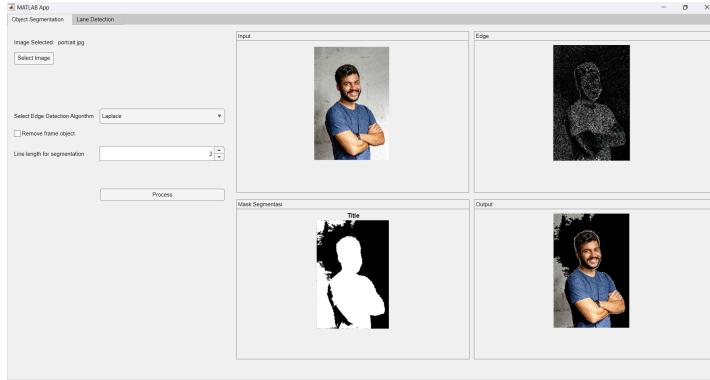
Gambar 2.9 Segmentasi objek dari citra kursi.jpg menggunakan Operator Laplace



Gambar 2.10 Segmentasi objek dari citra butterfly.jpg menggunakan Operator Laplace



Gambar 2.11 Segmentasi objek dari citra lotus.jpg menggunakan Operator Laplace



Gambar 2.12 Segmentasi objek dari citra portrait.jpg menggunakan Operator Laplace

Analisis:

Berdasarkan eksperimen yang telah dilakukan, Operator Laplace dapat digunakan untuk mendeteksi tepi. Tetapi, operator ini sensitif terhadap derau. Perbedaan yang signifikan pada *pixel* saat memasuki area derau dideteksi oleh operator ini sebagai tepi. Oleh karena itu, banyak *pixel* yang tidak termasuk objek, tetapi dimunculkan pada hasil segmentasi. Hal ini dapat dilihat pada mask segmentasi pada setiap citra uji di Gambar 2.1 - Gambar 2.12.

2. Operasi Laplace of Gaussian

Operasi Laplace of Gaussian (LoG) adalah operasi pengembangan dari Laplace. Pendekstian tepi dengan Operator Laplace terkadang menghasilkan tepi palsu yang disebabkan oleh derau. Untuk mengurangi kemunculan tepi palsu ini, citra ditapis terlebih dahulu dengan fungsi Gaussian. Operator ini dapat dilakukan dengan dua metode, yaitu

1. Konvolusi citra terhadap fungsi Gauss, kemudian lakukan Operasi Laplace terhadap hasilnya, atau
2. Konvolusi citra langsung dengan penapis LoG

Pada pengeraan ini, metode yang dilakukan adalah metode kedua. Fungsi LoG menerima parameter berupa standar deviasi. Ukuran mask konvolusi yang ditentukan adalah 6 dikali standar deviasi, di mana ini ditentukan dari rentang $[-3\sigma, +3\sigma]$, yang mencakup 99,7% dari distribusi Gaussian. Mask konvolusi dipastikan berupa bilangan ganjil. Rumus yang digunakan untuk menentukan mask konvolusi adalah

$$\nabla^2 G(x, y) = \frac{(x^2 + y^2 - 2\sigma^2)}{\sigma^4} e^{-\frac{(x^2+y^2)}{2\sigma^2}}$$

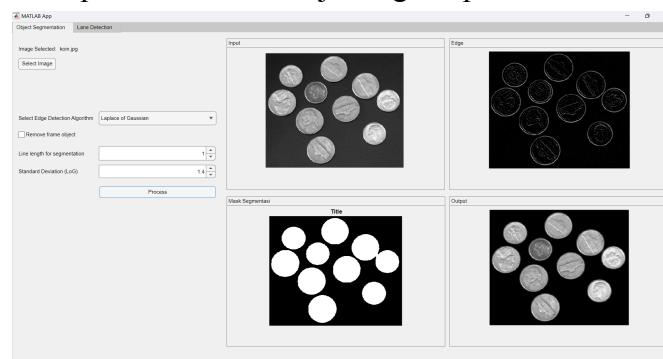
Dengan x dan y adalah koordinat pixel mask dan σ adalah standar deviasi. Berikut adalah kode implementasi Operasi Laplace of Gaussian

```
% laplaceOfGaussian.m
function outputImage = laplaceOfGaussian(image,sigma)
    % Kernel konvolusi citra
    sizeKernel = ceil(6 * sigma);
    disp(sizeKernel);
    % ukuran kernel harus ganjil
    if mod(sizeKernel, 2) == 0
        sizeKernel = sizeKernel + 1;
    end
    % membuat kernel untuk operasi LoG berdasarkan STD
    [x, y] = meshgrid(-floor(sizeKernel/2):floor(sizeKernel/2));
    % operasi LoG
    LoG = (x.^2 + y.^2 - 2*sigma^2) .* exp(-(x.^2 + y.^2) /
(2*sigma^2));
    LoG = LoG / (2 * pi * sigma^4); % Normalisasi kernel

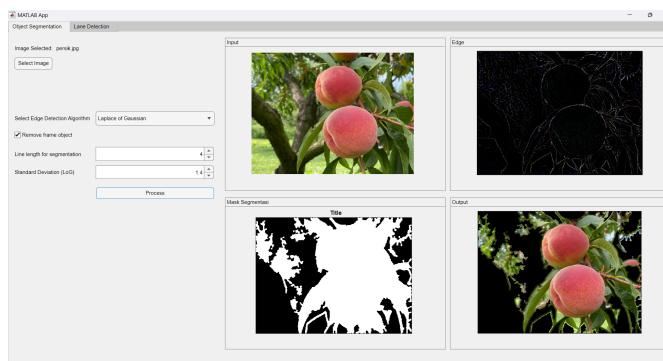
    % Proses konvolusi
    output = convolution(double(image), double(LoG));

    % Normalisasi citra
    outputImage = uint8(mat2gray(output) * 255);
end
```

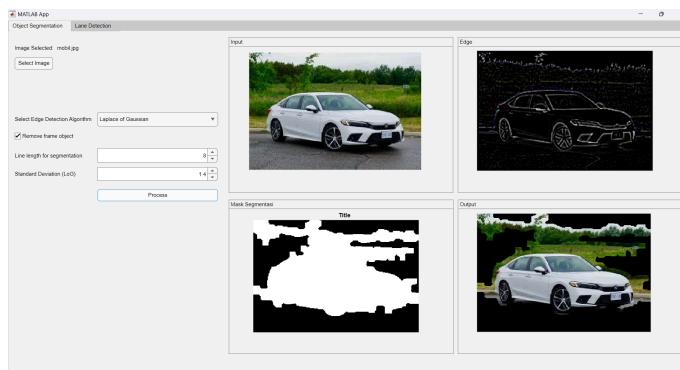
Berikut adalah hasil pemrosesan citra uji dengan operator LoG.



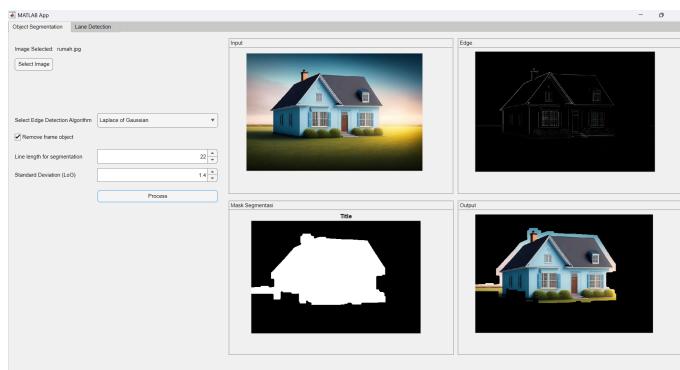
Gambar 2.13 Segmentasi objek dari citra koin.jpg menggunakan Operator LoG



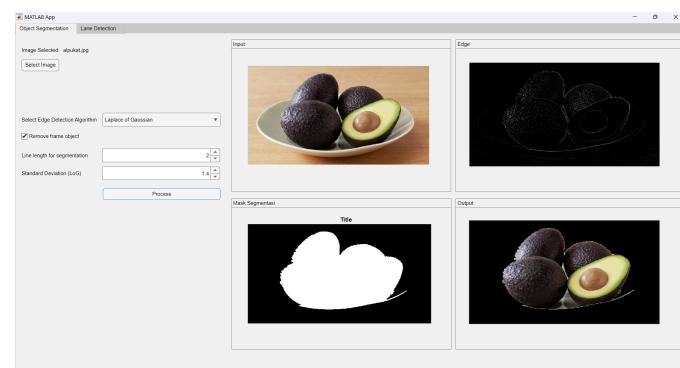
Gambar 2.14 Segmentasi objek dari citra persik.jpg menggunakan Operator LoG



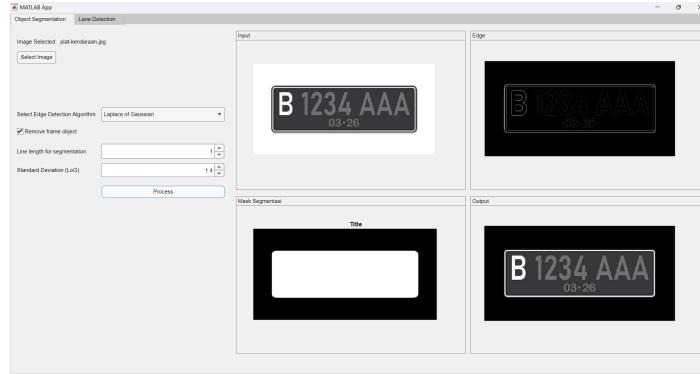
Gambar 2.15 Segmentasi objek dari citra mobil.jpg menggunakan Operator LoG



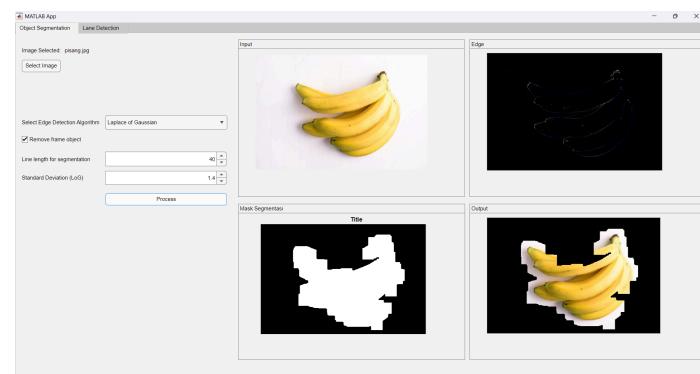
Gambar 2.16 Segmentasi objek dari citra rumah.jpg menggunakan Operator LoG



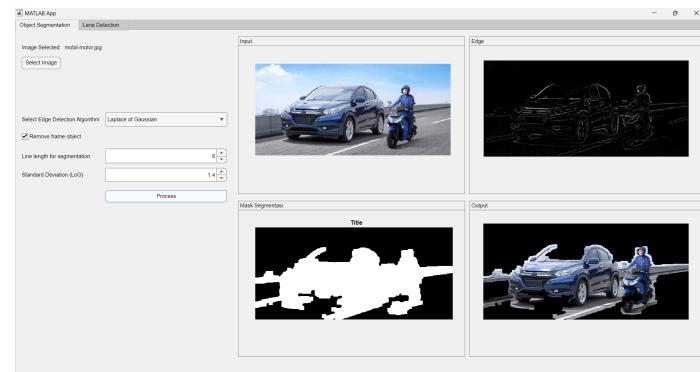
Gambar 2.17 Segmentasi objek dari citra alpukat.jpg menggunakan Operator LoG



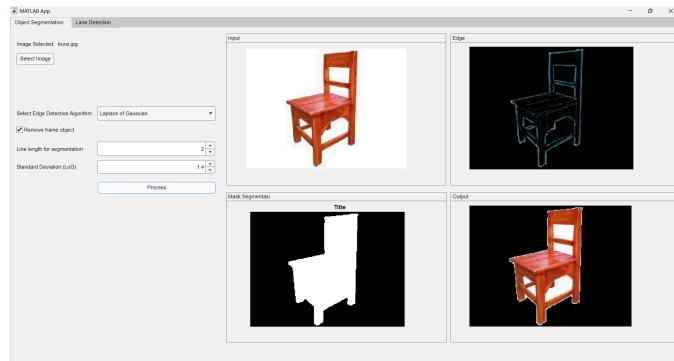
Gambar 2.18 Segmentasi objek dari citra mobil.jpg menggunakan Operator LoG



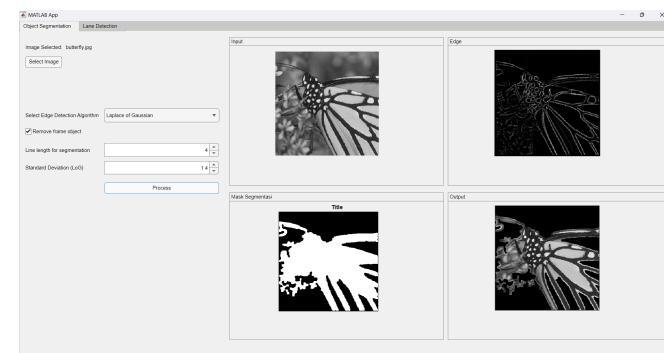
Gambar 2.19 Segmentasi objek dari citra pisang.jpg menggunakan Operator LoG



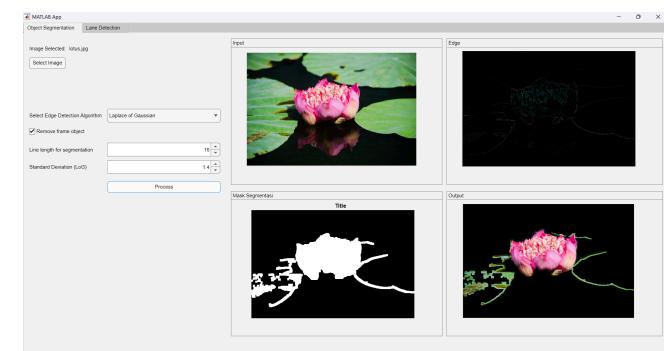
Gambar 2.20 Segmentasi objek dari citra mobil-motor.jpg menggunakan Operator LoG



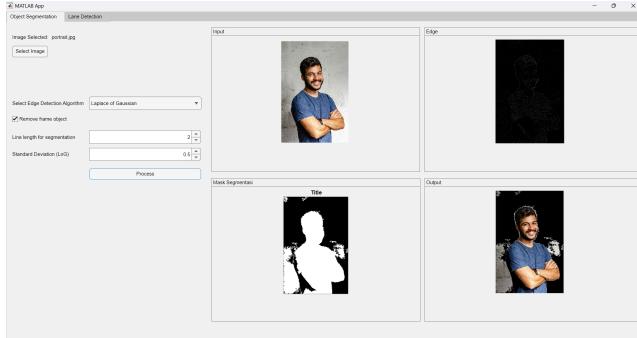
Gambar 2.21 Segmentasi objek dari citra kursi.jpg menggunakan Operator LoG



Gambar 2.22 Segmentasi objek dari citra butterfly.jpg menggunakan Operator LoG



Gambar 2.23 Segmentasi objek dari citra lotus.jpg menggunakan Operator LoG



Gambar 2.24 Segmentasi objek dari citra kursi.jpg menggunakan Operator LoG

Analisis:

Pengujian dilakukan dengan menggunakan simpangan baku sebesar 1.4. Dari hasil pengujian, dapat dilihat bahwa hasil segmentasi objek dengan operator LoG lebih baik daripada Laplace. Derau-derau yang dideteksi pada operator Laplace sudah berkurang saat diolah dengan operator LoG. Namun, panjang garis yang dibutuhkan untuk dilasi citra tepi cenderung lebih besar daripada Laplace. Hal ini disebabkan oleh penghalusan yang dilakukan oleh mask konvolusi LoG juga berdampak kepada detail-detail tepi.

3. Operasi Sobel

Operasi Sobel adalah salah satu operasi deteksi tepi yang memanfaatkan operator berupa magnitudo dari gradien yang dihitung dengan rumus:

$$M = \sqrt{s_x^2 + s_y^2}$$

Nilai s_x dan s_y diperoleh dari kombinasi piksel-piksel sekitar untuk setiap titik piksel (x, y) di dalam citra. Agar mempersingkat perhitungan, s_x dan s_y dapat diperoleh melalui konvolusi citra dengan 2 buah *mask* berukuran 3x3.

$$S_x = [[-1, 0, 1], [-2, 0, 2], [-1, 0, 1]]$$

$$S_y = [[1, 2, 1], [0, 0, 0], [-1, 2, -1]]$$

Berikut adalah kode implementasi operasi Sobel

```
% sobel_operator.m
function [result] = sobel_operator(I)
% Menyiapkan mask operator Sobel
mask_x = [-1,0,1; -2,0,2; -1,0,1];
mask_y = [1,2,1; 0,0,0; -1,-2,-1];

I = double(I);
[M, N, C] = size(I);
X = 3; % Ukuran mask
Y = 3; % Ukuran mask
% Menyiapkan matriks citra keluaran seukuran I
result = zeros(M, N, C);
```

```

% Iterasi untuk mengonvolusi citra dengan mask
for k = 1 : C
    for i = 1 : M - X + 1
        for j = 1 : N - Y + 1

            sx = 0; sy = 0;
            a = 0;
            for u = i : (i + X-1)
                a = a + 1;
                b = 0;
                for v = j : (j + Y-1)
                    b = b + 1;
                    sx = sx + mask_x(a,b) * I(u,v);
                    sy = sy + mask_y(a,b) * I(u,v);
                end
            end

            % Hitung magnitudo gradien untuk menggantikan nilai
            % tengah pada bagian citra yang tensorot oleh mask
            magnitude = floor(sqrt(sx^2 + sy^2));

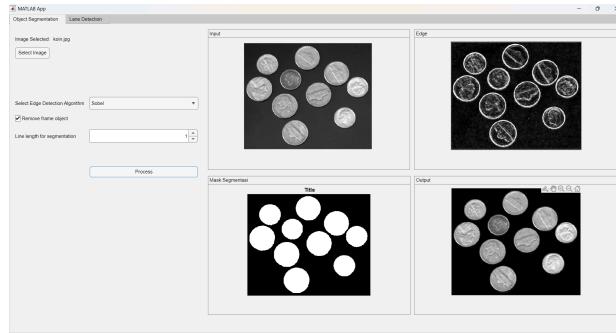
            % CLIPPING
            if (magnitude < 0)
                magnitude = 0;
            elseif (magnitude > 255)
                magnitude = 255;
            end

            % Mengganti nilai tengah yang tensorot mask
            result((i + floor(X/2)), (j + floor(Y/2)), k) = magnitude;
        end
    end
end
% Memasukkan bagian pinggir citra yang tidak ikut dikonvolusi
for i = 1 : M
    for j = 1 : N
        for k = 1 : C
            if (i <= floor(X/2) || j <= floor(Y/2) || i > M - floor(X/2)
|| j > N - floor(Y/2))
                result(i, j, k) = I(i, j, k);
            end
        end
    end
end

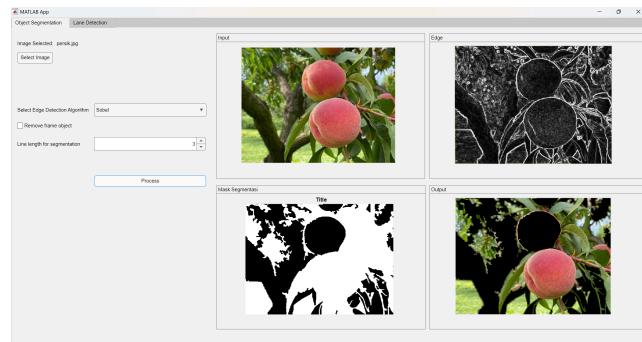
result = uint8(result);
end

```

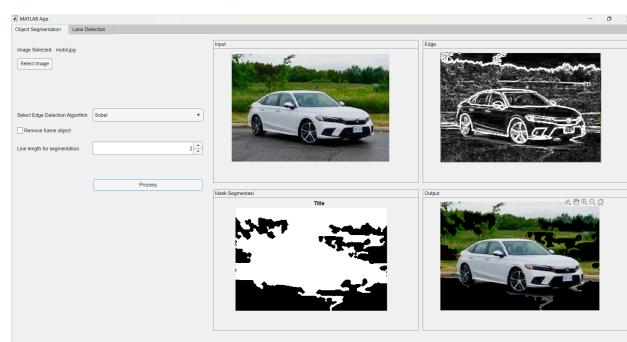
Berikut adalah hasil pengujian operasi Sobel terhadap beberapa citra uji.



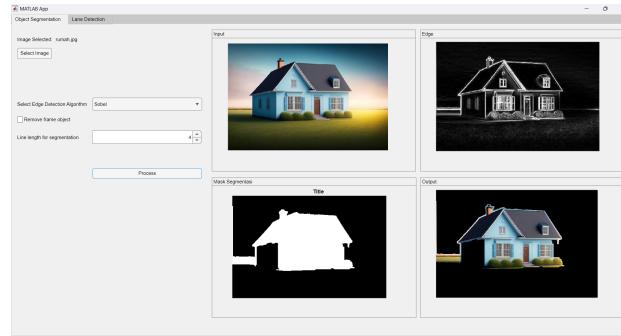
Gambar 2.25 Segmentasi objek dari citra koin.jpg menggunakan Operator Sobel



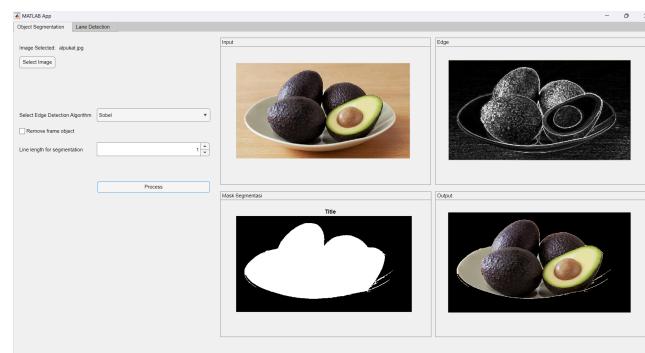
Gambar 2.26 Segmentasi objek dari citra persik.jpg menggunakan Operator Sobel



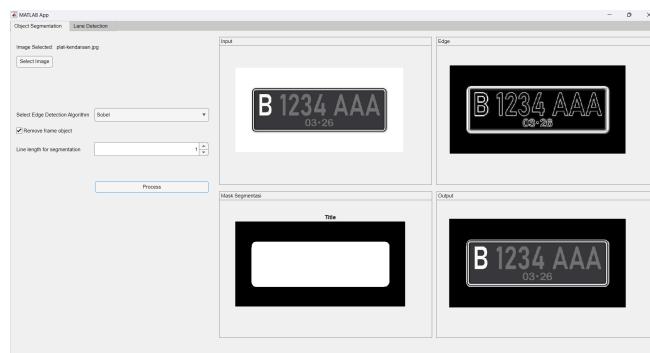
Gambar 2.27 Segmentasi objek dari citra mobil.jpg menggunakan Operator Sobel



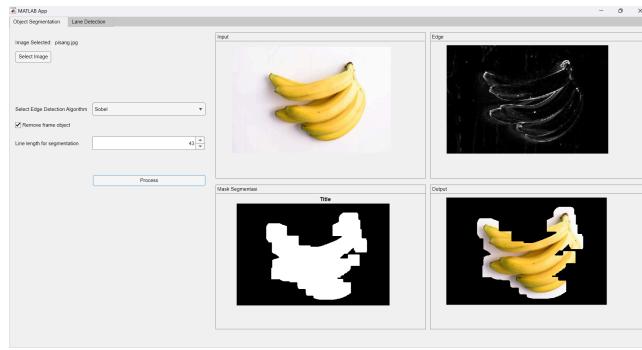
Gambar 2.28 Segmentasi objek dari citra rumah.jpg menggunakan Operator Sobel



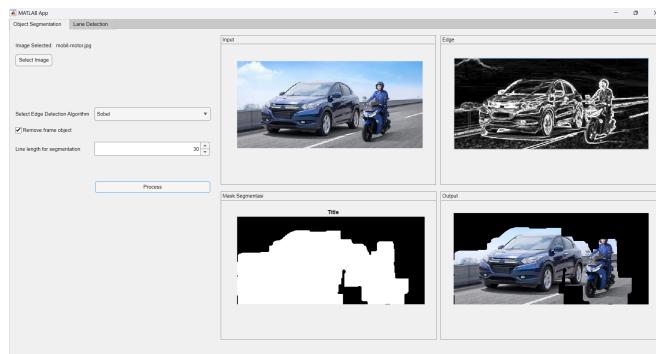
Gambar 2.29 Segmentasi objek dari citra alpukat.jpg menggunakan Operator Sobel



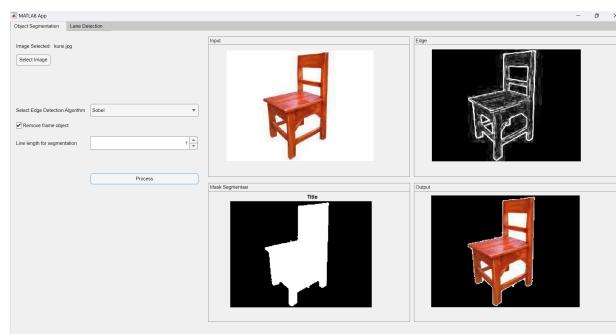
Gambar 2.30 Segmentasi objek dari citra plat-kendaraan.jpg menggunakan Operator Sobel



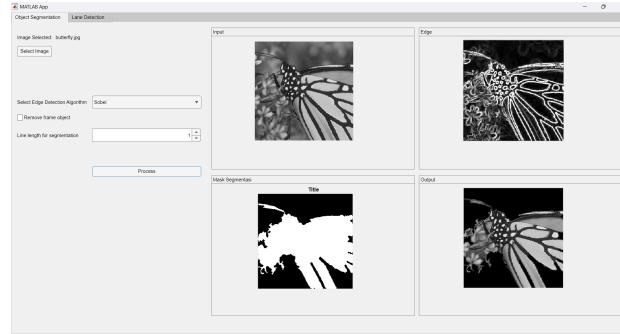
Gambar 2.31 Segmentasi objek dari citra pisang.jpg menggunakan Operator Sobel



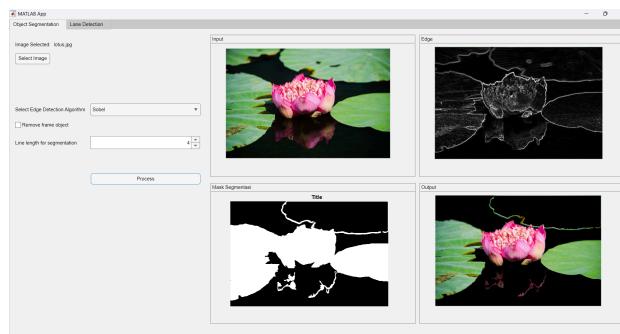
Gambar 2.32 Segmentasi objek dari citra mobil-motor.jpg menggunakan Operator Sobel



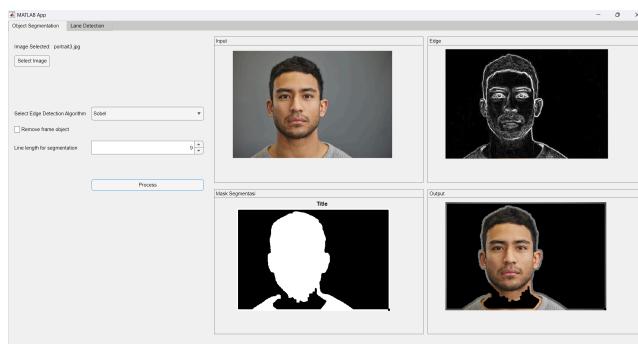
Gambar 2.33 Segmentasi objek dari citra kursi.jpg menggunakan Operator Sobel



Gambar 2.34 Segmentasi objek dari citra butterfly.jpg menggunakan Operator Sobel



Gambar 2.35 Segmentasi objek dari citra lotus.jpg menggunakan Operator Sobel



Gambar 2.36 Segmentasi objek dari citra portrait3.jpg menggunakan Operator Sobel

Analisis:

Dapat dilihat bahwa hasil deteksi tepi dengan operator Sobel memiliki ketebalan yang cukup tinggi. Hal ini membuat parameter line length untuk dilasi garis yang dibutuhkan oleh segmentasi objek menjadi lebih kecil, yaitu hanya pada rentang 1-5. Karena ketebalan tepinya juga, operasi ini menghasilkan segmentasi objek yang seolah-olah memiliki garis tepi yang mengelilingi objek. Deteksi tepi

dengan Sobel ini membutuhkan waktu yang cukup lama untuk citra beresolusi tinggi karena konvolusi dilakukan secara manual tanpa menggunakan *library*.

4. Operasi Prewitt

Mirip dengan operasi Sobel, deteksi tepi dengan operasi Prewitt juga memanfaatkan operator magnitudo gradien. Perbedaannya adalah nilai konstanta yang dipakai untuk mendapatkan hasil p_x dan p_y dari kombinasi piksel-piksel sekitar titik piksel (x, y) . Dengan demikian, *mask* konvolusi 3x3 yang digunakan untuk mendapatkan p_x dan p_y pun memiliki nilai yang berbeda.

$$M = \sqrt{p_x^2 + p_y^2}$$

$$P_x = [[-1, 0, 1], [-1, 0, 1], [-1, 0, 1]]$$

$$P_y = [[1, 1, 1], [0, 0, 0], [-1, -1, -1]]$$

Berikut adalah kode implementasi operasi Prewitt

```
% prewitt_operator.m
function [result] = prewitt_operator(I)
% Menyiapkan mask operator Prewitt
mask_x = [-1,0,1; -1,0,1; -1,0,1];
mask_y = [1,1,1; 0,0,0; -1,-1,-1];

I = double(I);
[M, N, C] = size(I);
X = 3; % Ukuran mask
Y = 3; % Ukuran mask
% Menyiapkan matriks citra keluaran seukuran I
result = zeros(M, N, C);

% Iterasi untuk mengonvolusi citra dengan mask
for k = 1 : C
    for i = 1 : M - X + 1
        for j = 1 : N - Y + 1

            sx = 0; sy = 0;
            a = 0;
            for u = i : (i + X-1)
                a = a + 1;
                b = 0;
                for v = j : (j + Y-1)
                    b = b + 1;
                    sx = sx + mask_x(a,b) * I(u,v);
                    sy = sy + mask_y(a,b) * I(u,v);
                end
            end

            % Hitung magnitudo gradien untuk menggantikan nilai
            % tengah pada bagian citra yang tersorot oleh mask
            magnitude = floor(sqrt(sx^2 + sy^2));

            % CLIPPING
            if (magnitude < 0)
                magnitude = 0;
            end
        end
    end
end
```

```

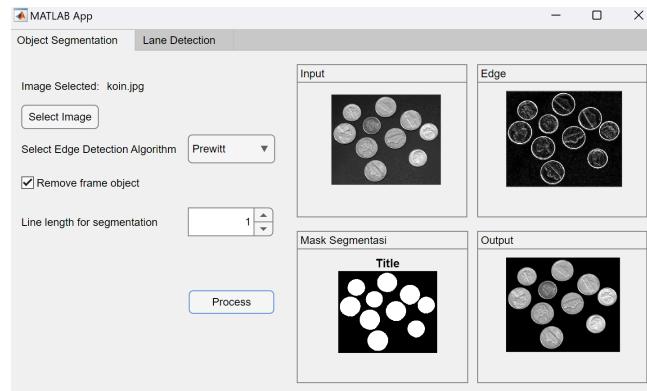
        elseif (magnitude > 255)
            magnitude = 255;
        end

        % Mengganti nilai tengah yang tersorot mask
        result((i + floor(X/2)), (j + floor(Y/2)), k) = magnitude;
    end
end
% Memasukkan bagian pinggir citra yang tidak ikut dikonvolusi
for i = 1 : M
    for j = 1 : N
        for k = 1 : C
            if (i <= floor(X/2) || j <= floor(Y/2) || i > M - floor(X/2)
|| j > N - floor(Y/2))
                result(i, j, k) = I(i, j, k);
            end
        end
    end
end

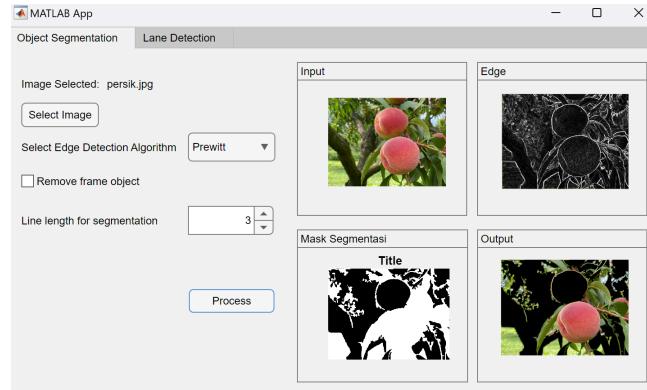
result = uint8(result);
end

```

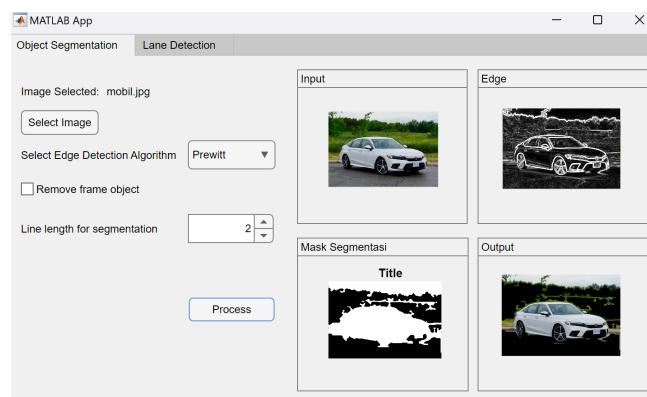
Berikut adalah hasil pemrosesan citra uji dengan operator Prewitt.



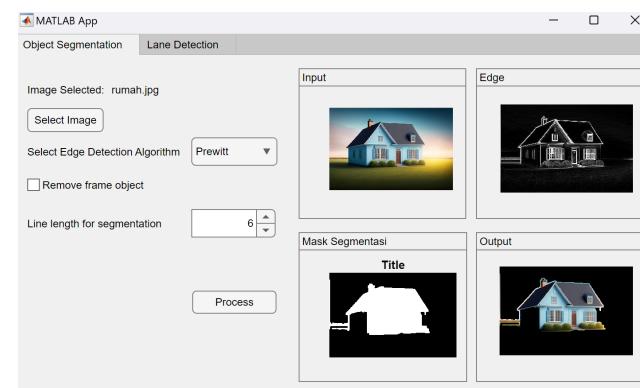
Gambar 2.37 Segmentasi objek dari citra koin.jpg menggunakan Operator Prewitt



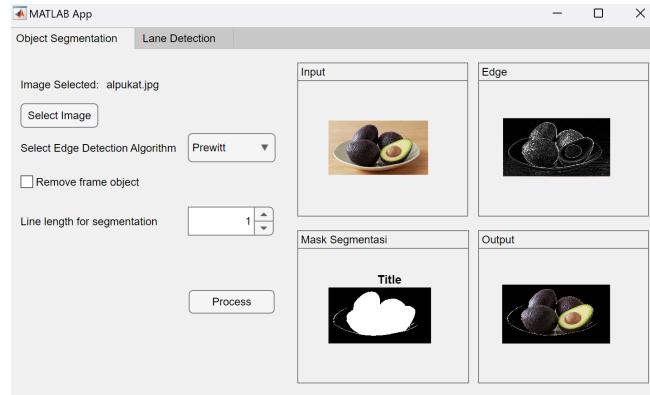
Gambar 2.38 Segmentasi objek dari citra persik.jpg menggunakan Operator Prewitt



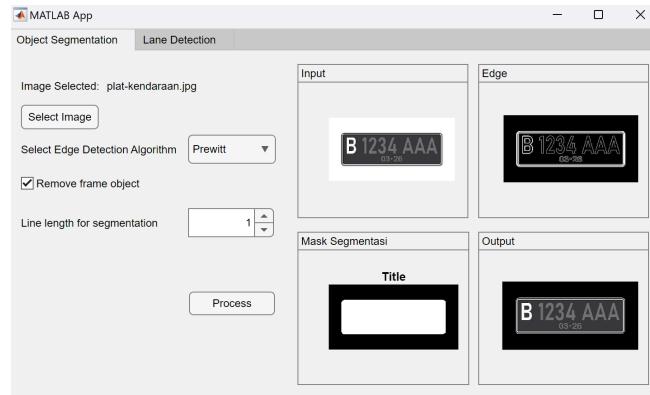
Gambar 2.39 Segmentasi objek dari citra mobil.jpg menggunakan Operator Prewitt



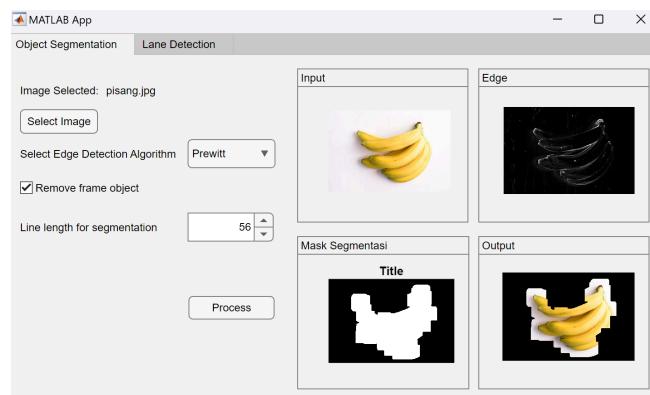
Gambar 2.40 Segmentasi objek dari citra rumah.jpg menggunakan Operator Prewitt



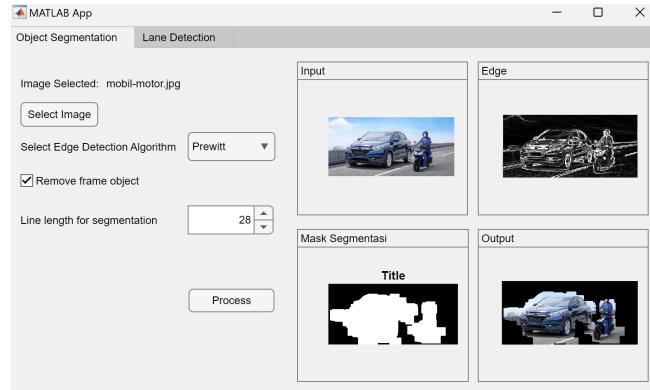
Gambar 2.41 Segmentasi objek dari citra alpukat.jpg menggunakan Operator Prewitt



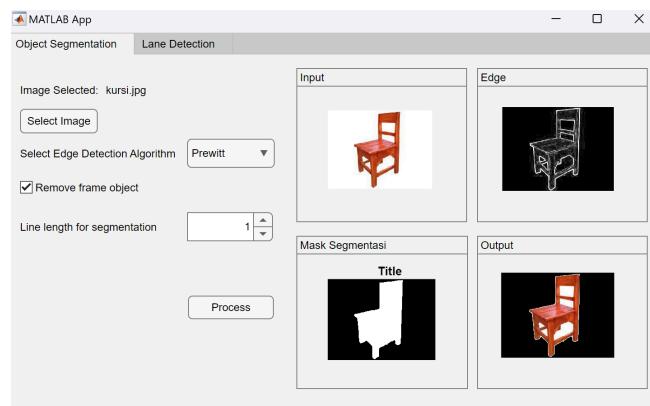
Gambar 2.42 Segmentasi objek dari citra plat-kendaraan.jpg menggunakan Operator Prewitt



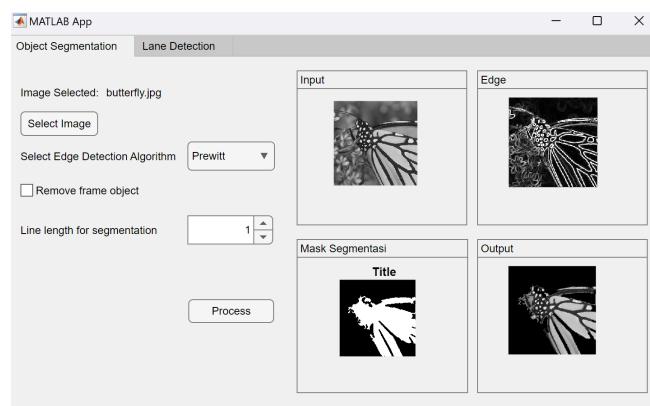
Gambar 2.43 Segmentasi objek dari citra pisang.jpg menggunakan Operator Prewitt



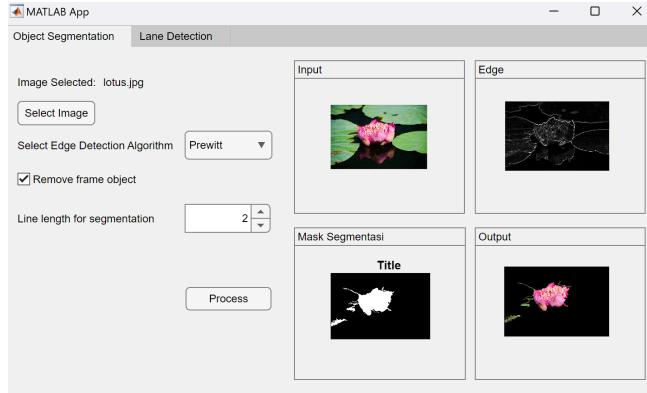
Gambar 2.44 Segmentasi objek dari citra mobil-motor.jpg menggunakan Operator Prewitt



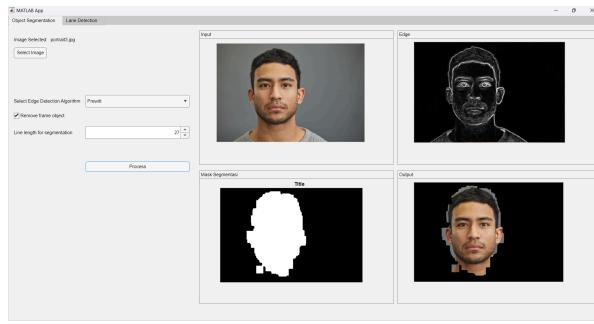
Gambar 2.45 Segmentasi objek dari citra kursi.jpg menggunakan Operator Prewitt



Gambar 2.46 Segmentasi objek dari citra butterfly.jpg menggunakan Operator Prewitt



Gambar 2.47 Segmentasi objek dari citra lotus.jpg menggunakan Operator Prewitt



Gambar 2.36 Segmentasi objek dari citra portrai3.jpg menggunakan Operator Prewitt

Analisis:

Operasi Prewitt menghasilkan tepi yang mirip dengan Sobel, namun dengan ketebalan tepi yang lebih tipis. Karena hal ini juga, Prewitt menghasilkan citra tepi yang kurang presisi karena tidak dapat mendeteksi tepi halus. Operator ini juga rentan terhadap kehilangan detail tepi, sehingga dibutuhkan parameter line length yang lebih besar daripada Sobel untuk dilasi garis dalam segmentasi objek dengan operator ini.

5. Operasi Roberts

Operasi Roberts adalah operasi pendekripsi citra yang memanfaatkan gradien dalam arah-x dan arah-y. Gradien ini dihitung dengan rumus

$$R_+(x, y) = f(x + 1, y + 1) - f(x, y)$$

$$R_-(x, y) = f(x, y + 1) - f(x + 1, y)$$

Dalam bentuk mask konvolusi, persamaan tersebut dapat dituliskan sebagai

$$R_+ = [[1 \ 0][0 \ -1]]$$

$$R_- = [[0 \ 1][-1 \ 0]]$$

Berikut adalah kode implementasi Operasi Roberts

```
% roberts.m
function outputImage = roberts(image)
    image = double(image);
    [rows, cols, colorChannel] = size(image);

    % Kernel X dan Y untuk algoritma roberts
    Rx = [1 0; 0 -1]; % Horizontal gradient
    Ry = [0 1; -1 0]; % Vertical gradient

    % matriks output
    Jx = zeros(rows, cols, colorChannel);
    Jy = zeros(rows, cols, colorChannel);

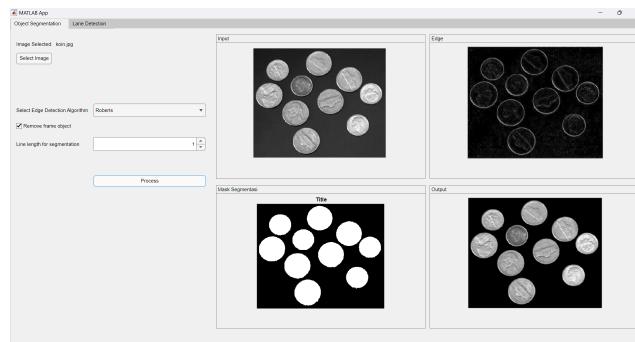
    % Konvolusi citra terhadap masing-masing kernel
    for k = 1:colorChannel
        for i = 1:(rows-1)
            for j = 1:(cols-1)
                % Mengambil submatrix berukuran 2x2
                subMatrix = image(i:i+1, j:j+1);

                % Kalikan dengan kernel
                Jx(i, j, k) = sum(sum(subMatrix .* Rx));
                Jy(i, j, k) = sum(sum(subMatrix .* Ry));
            end
        end
    end

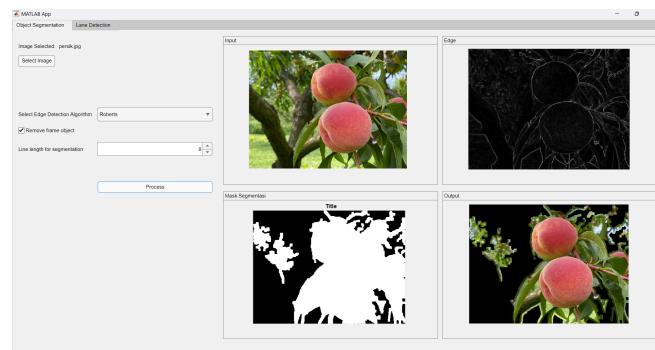
    % Menghitung Jedge
    Jedge = sqrt(Jx.^2 + Jy.^2);

    % Normalize gradient magnitude to range 0-255
    outputImage = uint8(mat2gray(Jedge) * 255);
end
```

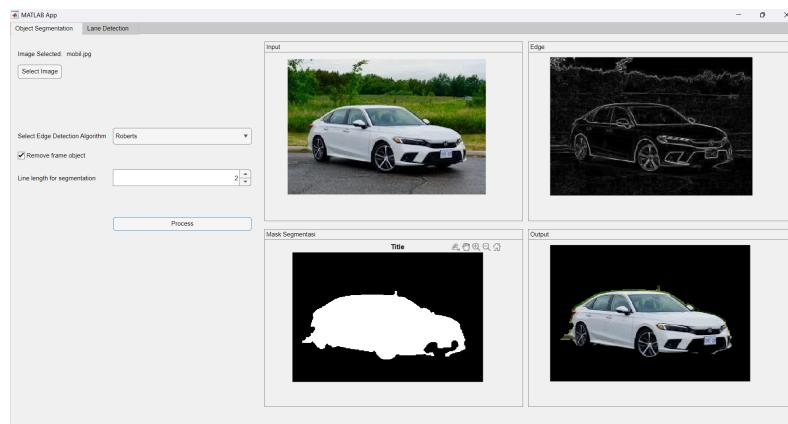
Berikut adalah hasil pengujian citra uji dengan operator Roberts.



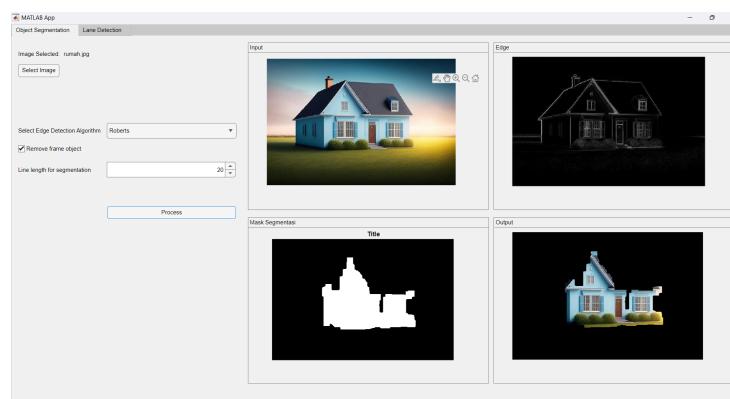
Gambar 2.49 Segmentasi objek dari citra koin.jpg menggunakan Operator Roberts



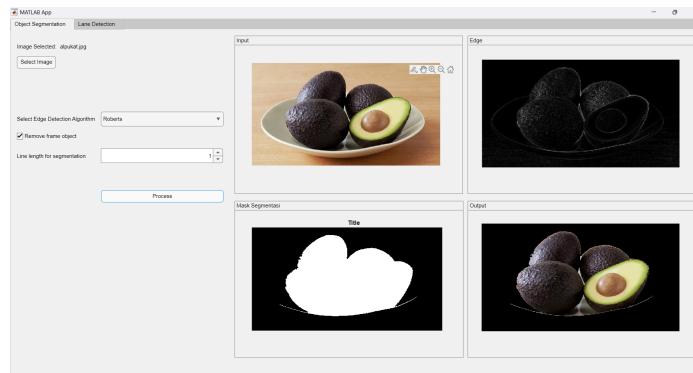
Gambar 2.50 Segmentasi objek dari citra persik.jpg menggunakan Operator Roberts



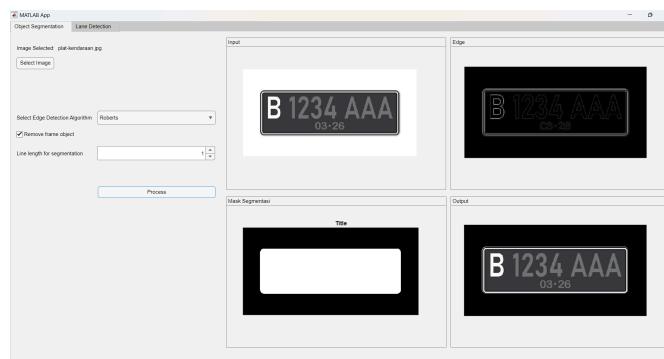
Gambar 2.51 Segmentasi objek dari citra mobil.jpg menggunakan Operator Roberts



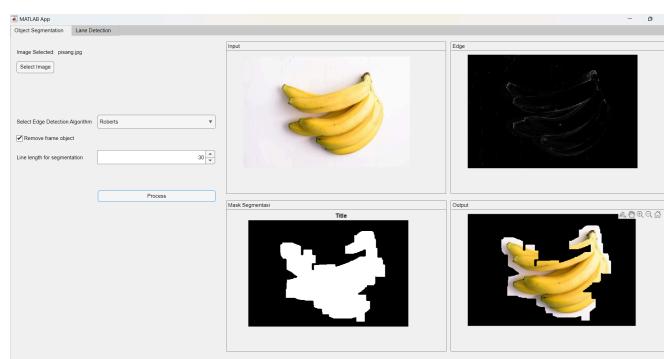
Gambar 2.52 Segmentasi objek dari citra rumah.jpg menggunakan Operator Roberts



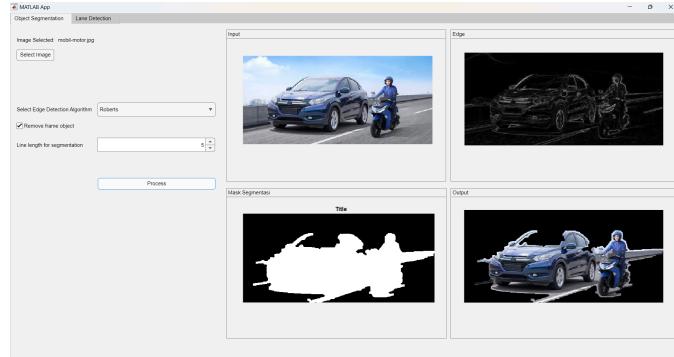
Gambar 2.53 Segmentasi objek dari citra alpukat.jpg menggunakan Operator Roberts



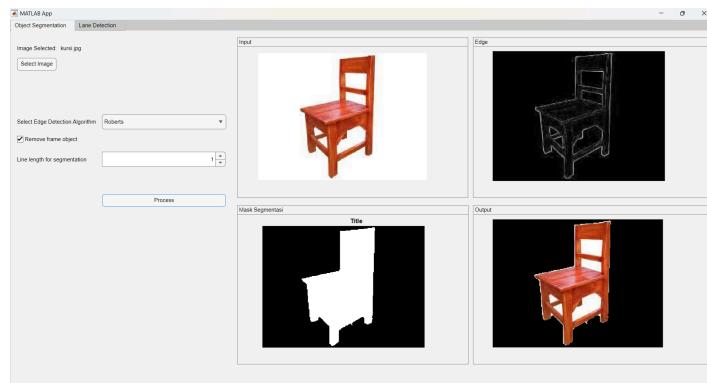
Gambar 2.54 Segmentasi objek dari citra plat-kendaraan.jpg menggunakan Operator Roberts



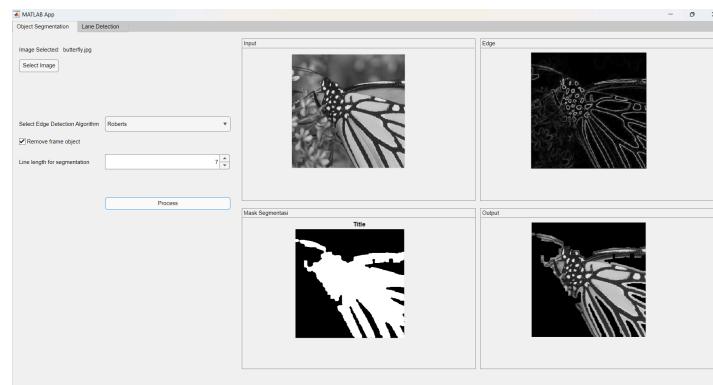
Gambar 2.55 Segmentasi objek dari citra pisang.jpg menggunakan Operator Roberts



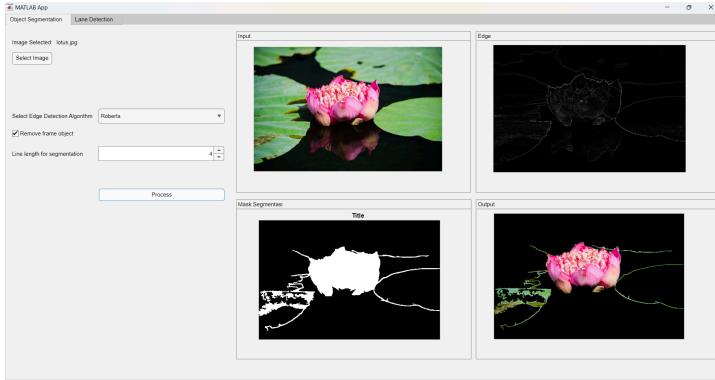
Gambar 2.56 Segmentasi objek dari citra mobil-motor.jpg menggunakan Operator Roberts



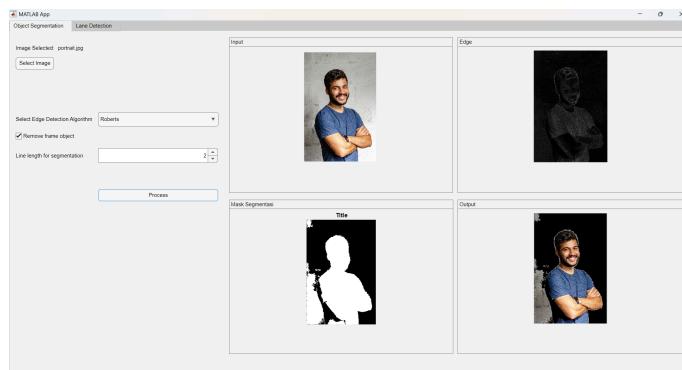
Gambar 2.57 Segmentasi objek dari citra kursi.jpg menggunakan Operator Roberts



Gambar 2.58 Segmentasi objek dari citra butterfly.jpg menggunakan Operator Roberts



Gambar 2.59 Segmentasi objek dari citra lotus.jpg menggunakan Operator Roberts



Gambar 2.60 Segmentasi objek dari citra portrait.jpg menggunakan Operator Roberts

Analisis:

Jika dibandingkan dengan hasil Sobel dan Prewitt, operasi ini menghasilkan garis tepi yang lebih tipis. Operasi ini memberikan hasil citra tepi yang lebih presisi dibanding Sobel dan Prewitt. Operasi ini memiliki kernel/mask yang lebih kecil daripada Sobel dan Prewitt, sehingga hanya dapat mendeteksi perubahan intensitas secara lokal. Tetapi hal ini membuat operasi Robert membutuhkan waktu pemrosesan yang lebih sedikit dibandingkan Sobel dan Prewitt.

6. Operasi Canny

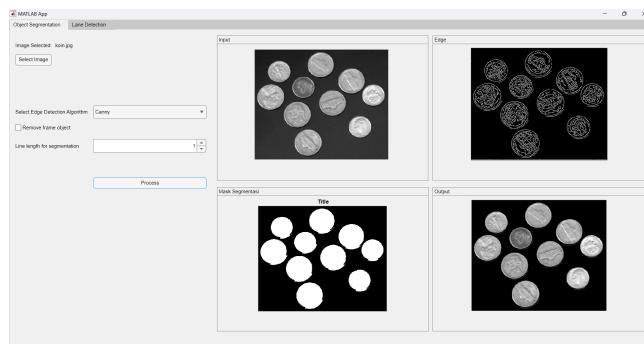
Operasi Canny adalah salah satu operasi deteksi tepi yang terkenal karena dapat menghasilkan tepi dengan ketebalan satu piksel. Pertama-tama, citra asli dilewatkan pada penapis Gaussian untuk menghaluskan citra sekaligus menghilangkan derau. Kemudian, gradien dihitung operator Sobel, Prewitt, atau Roberts. Operator Canny sendiri adalah pengambangan (*thresholding*) sebagai lanjutan operator-operator sebelumnya.

Terdapat dua nilai ambang yang digunakan, yaitu T1 dan T2 ($T1 < T2$). Jika magnitudo gradien lebih besar dari T2, piksel tersebut adalah tepi kuat (*strong edge*). Setiap piksel yang terhubung dengan tepi kuat dan memiliki magnitudo gradien lebih besar dari T1, adalah tepi lemah (*weak edge*). Terakhir, dilakukan penautan tepi (*edge-linking*) untuk menghubungkan tepi lemah dengan tepi kuat.

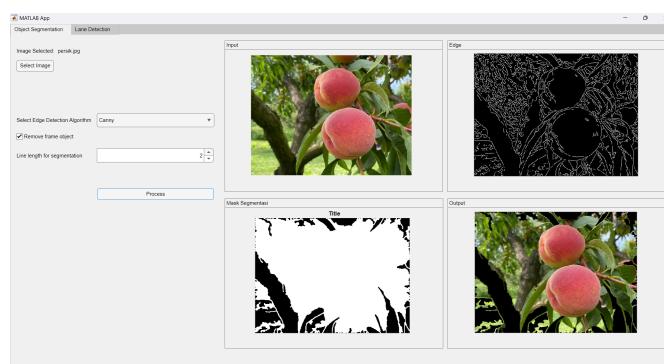
Meskipun operasinya terkesan rumit, Matlab telah menyediakan fungsi `edge` untuk mendeteksi tepi secara langsung dengan pilihan berbagai metode, tidak terkecuali Canny. Berikut adalah kode implementasi operasi Canny.

```
% canny_operator.m
function [result] = canny_operator(I)
    % Mengonversi citra ke grayscale
    if (size(I,3) == 3)
        I = rgb2gray(I);
    end
    % Fungsi bawaan operator Canny
    result = edge(I, 'Canny');
end
```

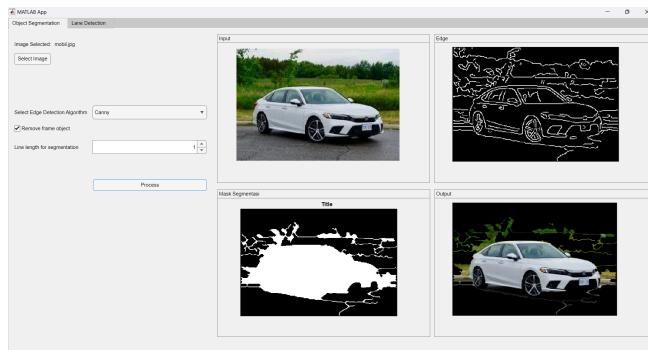
Berikut adalah hasil pemrosesan citra uji dengan operator Canny.



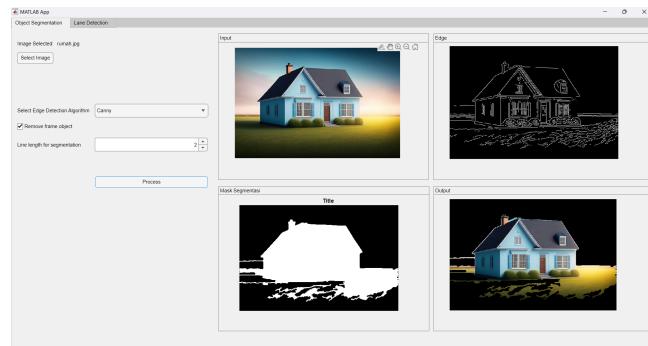
Gambar 2.61 Segmentasi objek dari citra koin.jpg menggunakan Operator Canny



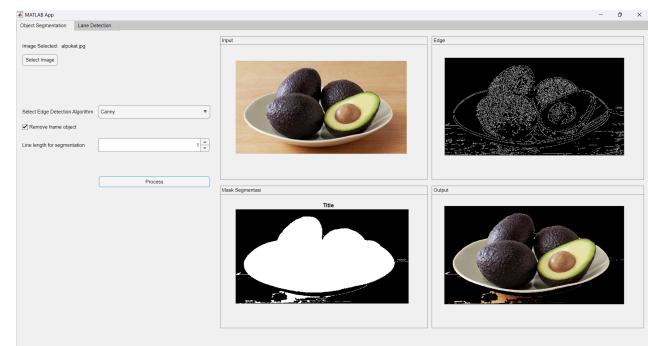
Gambar 2.62 Segmentasi objek dari citra persik.jpg menggunakan Operator Canny



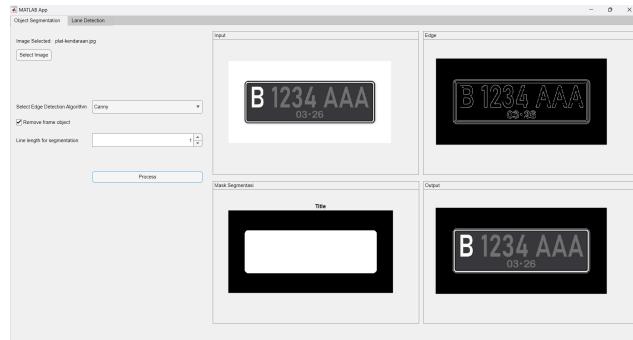
Gambar 2.63 Segmentasi objek dari citra mobil.jpg menggunakan Operator Canny



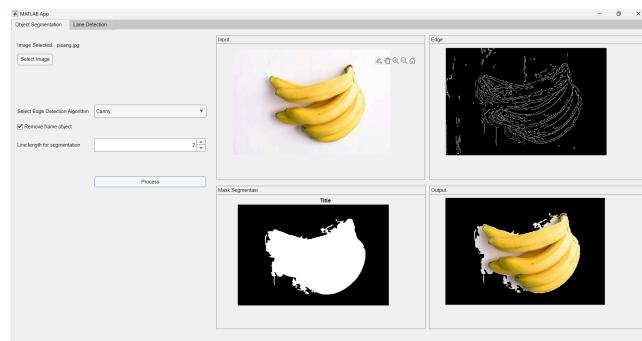
Gambar 2.64 Segmentasi objek dari citra rumah.jpg menggunakan Operator Canny



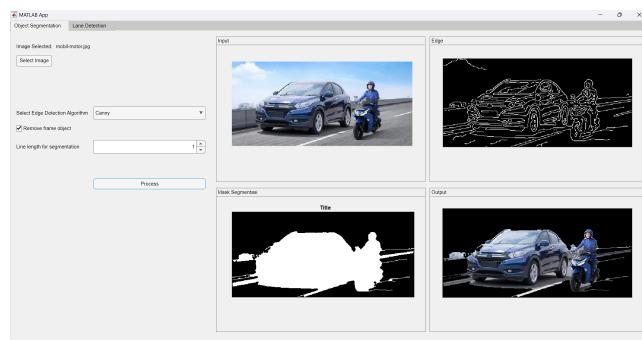
Gambar 2.65 Segmentasi objek dari citra alpukat.jpg menggunakan Operator Canny



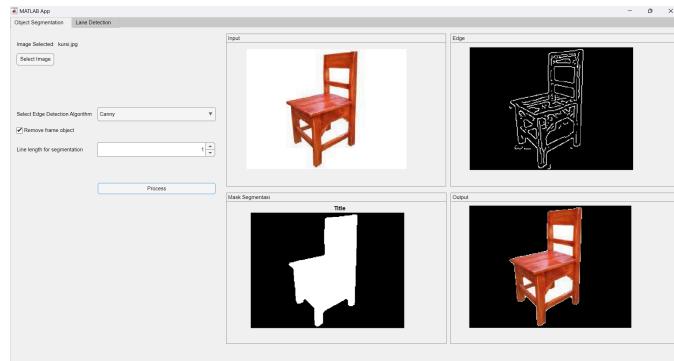
Gambar 2.66 Segmentasi objek dari citra plat-kendaraan.jpg menggunakan Operator Canny



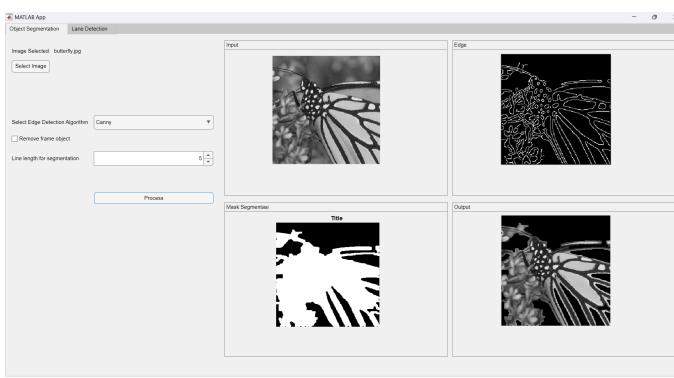
Gambar 2.67 Segmentasi objek dari citra pisang.jpg menggunakan Operator Canny



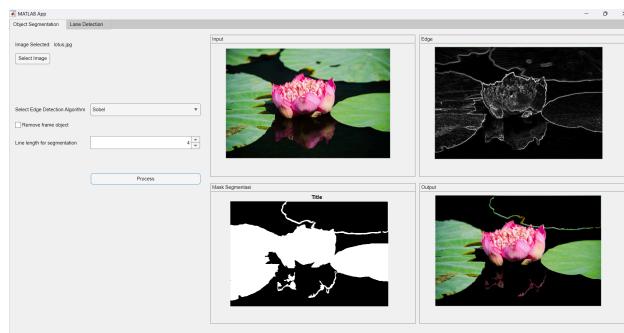
Gambar 2.68 Segmentasi objek dari citra mobil-motor.jpg menggunakan Operator Canny



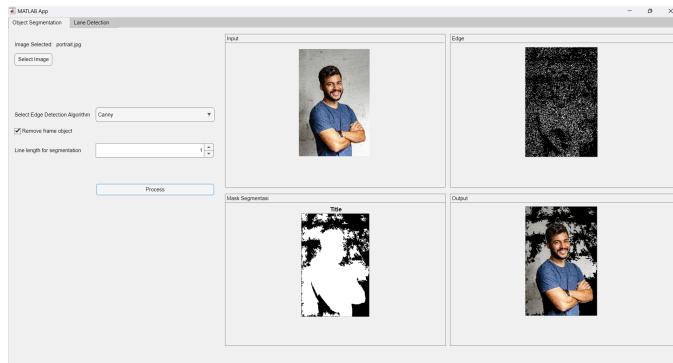
Gambar 2.69 Segmentasi objek dari citra kursi.jpg menggunakan Operator Canny



Gambar 2.70 Segmentasi objek dari citra butterfly.jpg menggunakan Operator Canny



Gambar 2.71 Segmentasi objek dari citra lotus.jpg menggunakan Operator Canny



Gambar 2.72 Segmentasi objek dari citra *portrait.jpg* menggunakan Operator Canny

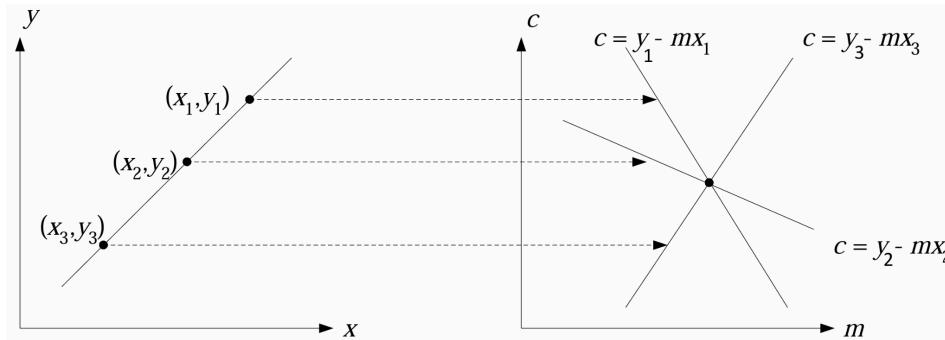
Analisis:

Secara umum, hasil segmentasi berdasarkan citra tepi hasil operator Canny tidak begitu baik dibandingkan hasil segmentasi berdasarkan citra tepi hasil operator Sobel, Prewitt, maupun Roberts. Pada beberapa kasus seperti citra pisang dan mobil-motor, performa segmentasi ini memang sangat memuaskan. Akan tetapi, performanya sangat buruk ketika berurusan dengan citra persik atau *portrait*. Hal ini bisa terjadi karena citra tepi memiliki tepi lemah dan kuat, sehingga sebagian objek pada latar malah ikut tersegmentasi.

III. Deteksi Jalur Garis

Pendeteksian jalur garis pada jalan raya dilakukan dengan bantuan transformasi Hough. Transformasi Hough adalah teknik untuk menghampiri kurva teratur dalam bentuk parametrik (kurva berupa garis, lingkaran, elips, dan sebagainya). Transformasi ini menspesifikasi kurva dalam bentuk parametrik $(x(u), y(u))$ dari parameter u .

Pertama-tama, citra tepi dihasilkan dengan operasi Canny. Kemudian, persamaan garis lurus $y = mx + c$ diubah menjadi $c = y - mx$. Jadi, setiap titik (x, y) pada citra tepi berkorespondensi dengan sebuah garis lurus dalam ruang parameter m, c . Keberadaan garis lurus pada citra ditandai dengan penumpukan suara pada tempat-tempat tertentu di dalam ruang parameter tersebut, yang diketahui dengan melakukan “pemungutan suara”.

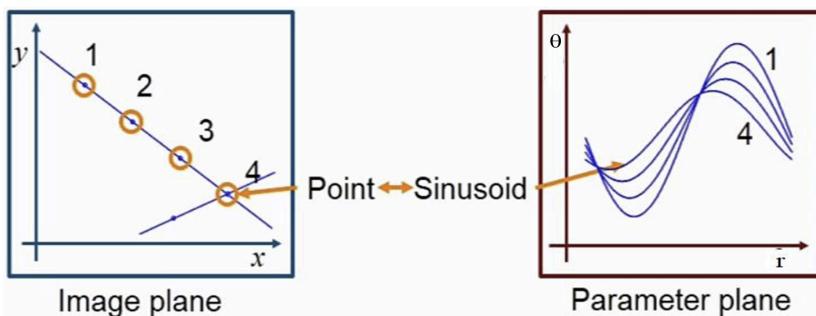


Sumber: Munir, R. (2024). Pendektsian Tepi (Bagian 3); bahan ajar mata kuliah IF4073 Pemrosesan Citra Digital, Institut Teknologi Bandung (diakses pada tanggal 30 November 2024 dari <https://informatika.stei.itb.ac.id/~rinaldi.munir/Citra/2024-2025/20-Pendeteksian-Tepi-Bagian3-2024.pdf>

Bagaimanapun juga, model parametrik di atas tidak dapat digunakan untuk mendeteksi garis vertikal karena nilai gradiennya tak berhingga. Oleh karena itu, garis dinyatakan dalam representasi polar, melibatkan parameter jarak (ρ atau r), serta sudut (θ). Persamaannya menjadi $r = x \cos \theta + y \sin \theta$. Perlu dicatat bahwa:

$$-\sqrt{M^2 + N^2} \leq r \leq \sqrt{M^2 + N^2}$$

$$-\frac{\pi}{2} \leq \theta \leq \frac{\pi}{2}$$



Sumber: Munir, R. (2024). Pendektsian Tepi (Bagian 3); bahan ajar mata kuliah IF4073 Pemrosesan Citra Digital, Institut Teknologi Bandung (diakses pada tanggal 30 November 2024 dari <https://informatika.stei.itb.ac.id/~rinaldi.munir/Citra/2024-2025/20-Pendeteksian-Tepi-Bagian3-2024.pdf>

Operasi pengambangan (*thresholding*) dilakukan terhadap citra H , yaitu citra hasil transformasi Hough. Titik-titik pada H yang memiliki nilai lebih dari batas T yang ditentukan (artinya mendapat *vote* lebih dari T kali pada pemungutan suara) akan diubah nilainya menjadi 1, dan selebihnya menjadi 0. Terakhir, dilakukan transformasi Hough balikan terhadap H untuk mendapatkan citra berisi garis-garis yang terdeteksi dalam citra semula.

Implementasi deteksi garis dalam Matlab melibatkan beberapa fungsi pembantu yang masing-masing dibuat sendiri, sebagai berikut.

<pre>% lookup_table.m function [COS, SIN] = lookup_table(m) COS = zeros(1, m); SIN = zeros(1, m); for i = 1 : m th = i * pi/(m-1) - pi/2; COS(i) = double(cos(double(th))); SIN(i) = double(sin(double(th))); end end</pre>	<p>Penjelasan:</p> <p>Membuat tabel <i>SIN</i> dan <i>COS</i> yang berisi nilai <i>sinus</i> serta <i>cosinus</i> untuk setiap sudut pada rentang yang digunakan</p>
<pre>% hough_transform.m function [P] = hough_transform(edge_I, q, p) [M, N] = size(edge_I); SQRTD = sqrt(M^2 + N^2); p = floor(p + 2*SQRTD); P = zeros(p, q); [COS, SIN] = lookup_table(q); for k = 1 : M for l = 1 : N if edge_I(k, l) == 1 for i = 1 : q r = l * COS(i) + k * SIN(i); r = (r + SQRTD) / (2 * SQRTD) * (p - 1) + 0.5; j = floor(r); if j >= 1 && j <= p % Pemungutan suara P(j, i) = P(j, i) + 1; end end end end end</pre>	<p>Penjelasan:</p> <p>Melakukan transformasi Hough, dengan cara setiap piksel <i>edge</i> yang terdeteksi menyumbangkan suara untuk titik-titik tertentu (titik-titik yang membentuk garisnya) dalam ruang parameter m,c</p>
<pre>% threshold.m function [P] = threshold(P, T) [p, q] = size(P); for i = 1 : p for j = 1 : q if P(i, j) > T P(i, j) = 1; end end end</pre>	<p>Penjelasan:</p> <p>Melakukan operasi pengambangan (<i>thresholding</i>) agar setiap titik yang bernilai lebih dari T</p>

<pre> else P(i, j) = 0; end end end </pre>	<p>pada hasil transformasi Hough diubah nilainya menjadi 1, dan selebihnya jadi 0</p>
<pre> % inverse_hough_transform.m function [out] = inverse_hough_transform(edge_I, P) [M, N] = size(edge_I); [p, q] = size(P); [COS, SIN] = lookup_table(q); out = zeros(M, N, 3); SQRTD = sqrt(M^2 + N^2); for k = 1 : q for l = 1 : p y = 0; x = 0; if P(l, k) == 1 for i = 1 : N r = (l * 2.0 * SQRTD / (p - 1)) - SQRTD; if SIN(k) == 0 y = y + 1; else y = (r - (i * COS(k))) / SIN(k); end y = y + 0.5; j = floor(y); if j >= 1 && j <= M out(j, i, 1) = 1; end end for j = 1 : M r = (l * 2.0 * SQRTD / (p - 1)) - SQRTD; if COS(k) == 0 x = x + 1; else x = (r - (j * SIN(k))) / COS(k); end x = x + 0.5; i = floor(x); if i >= 1 && i <= N out(j, i, 1) = 1; end end end end end end </pre>	<p>Penjelasan:</p> <p>Melakukan transformasi Hough balikan agar titik-titik yang telah diubah nilainya menjadi 1 pada hasil transformasi Hough tampil sebagai garis yang terdeteksi pada citra semula.</p> <p>Dalam iterasi nilai ρ (q) dan θ (p), terdapat nilai terdapat dua buah <i>loop</i> untuk menggambar garis, yaitu dalam arah sumbu horizontal dan vertikal. Hal ini bertujuan untuk memantapkan garis yang dihasilkan.</p>
<pre> % line_detection.m function [H, theta, rho, HT, lines, final] = line_detection(I, T) </pre>	<p>Penjelasan:</p>

```

% Deteksi tepi dengan operator Canny
[result] = canny_operator(I);

% Transformasi Hough terhadap citra tepi
H = hough_transform(result, 180, 0);
% Mendaftarkan theta dan rho yang digunakan sebagai parameter
% justifikasi penampilan gambar dengan fungsi imshowHough
theta = linspace(-90, 89, 180);
[M, N] = size(I);
SQRTD = sqrt(M^2 + N^2);
rho = linspace(-SQRTD, SQRTD, 2*SQRTD);

% Pengambangan terhadap hasil transformasi Hough
HT = threshold(H, T);

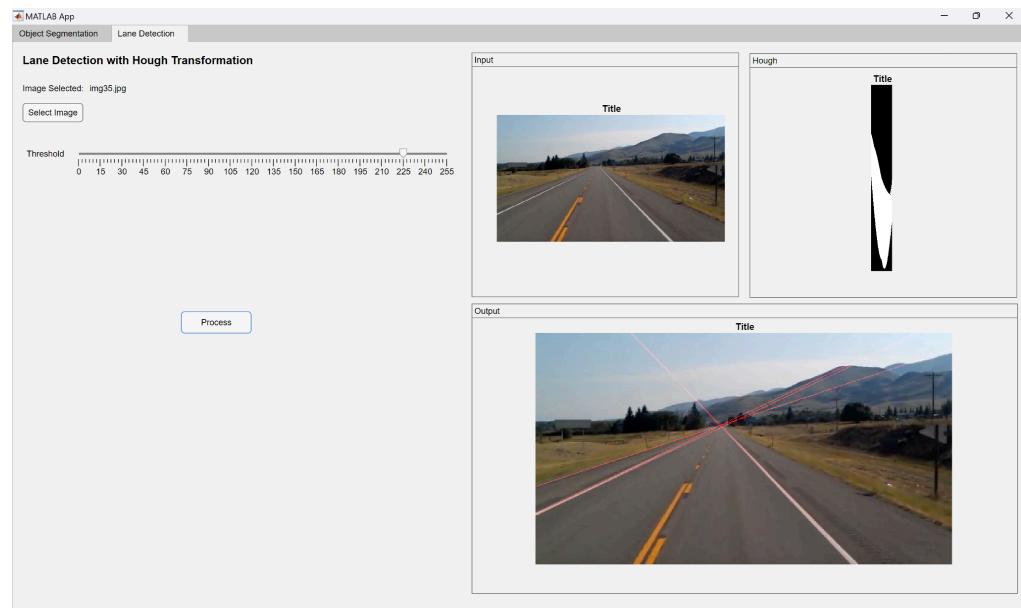
% Transformasi Hough balikan terhadap hasil pengambangan
% Hasil: garis-garis yang terdeteksi pada citra tepi
lines = inverse_hough_transform(result, HT);

% Menambahkan garis-garis pada citra semula
final = im2double(I) + lines;
end

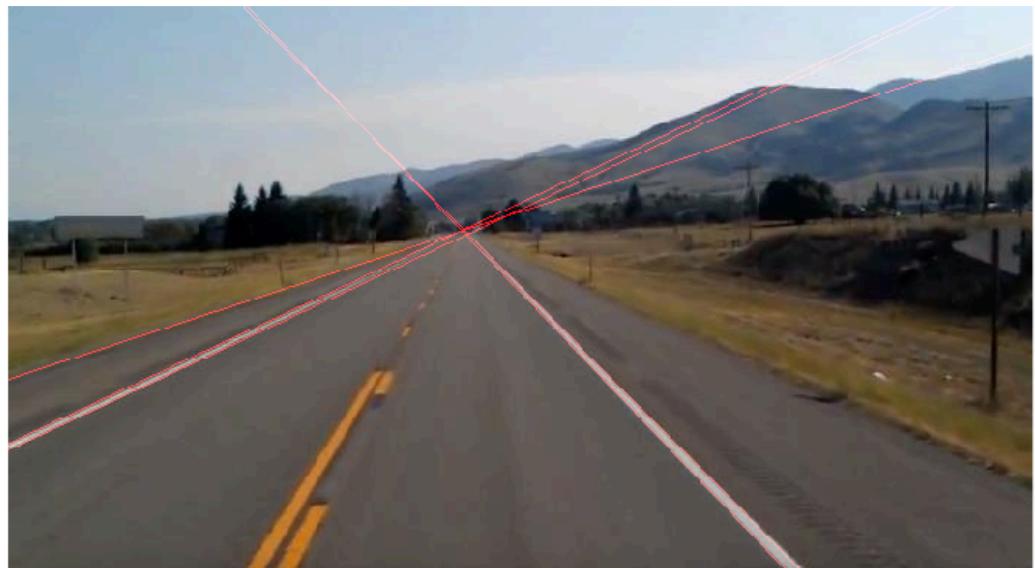
```

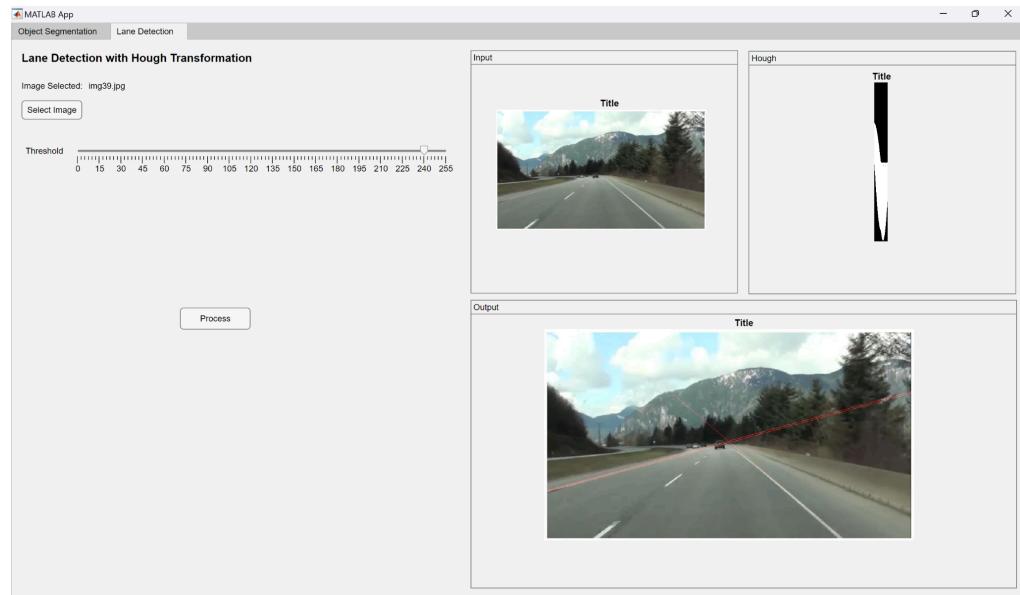
Menggabungkan semua fungsi yang dibuat untuk deteksi garis dengan transformasi Hough.

`final` merupakan citra semula yang ditambahkan dengan citra garis yang terdeteksi, sehingga dapat ditampilkan sebagai *output* yang penuh semantik.

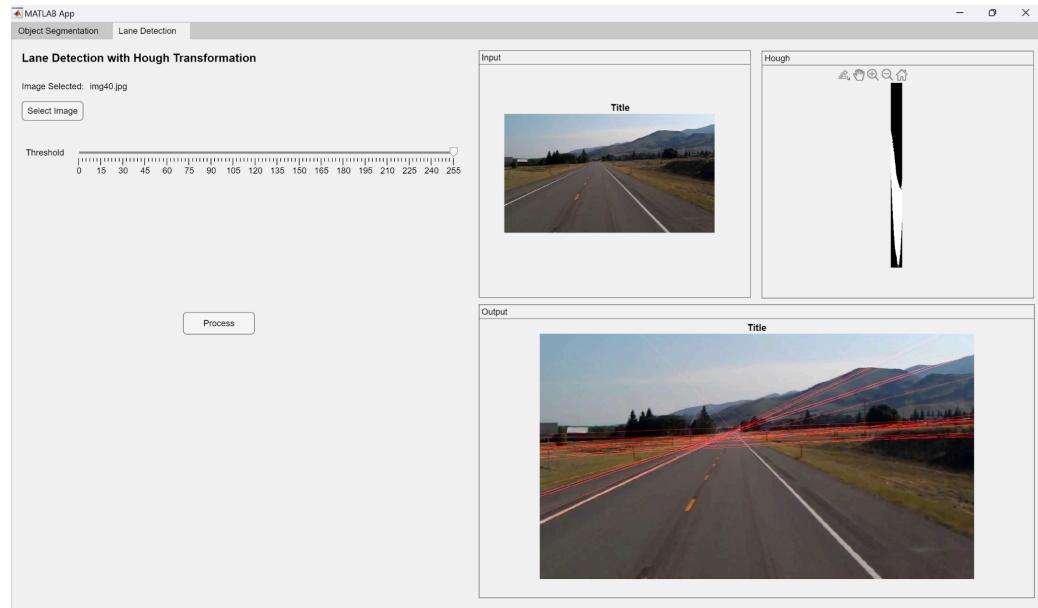


Gambar 3.1 Deteksi Garis Jalur dari Citra img35.jpg
dengan Bantuan Transformasi Hough ($T=225$)

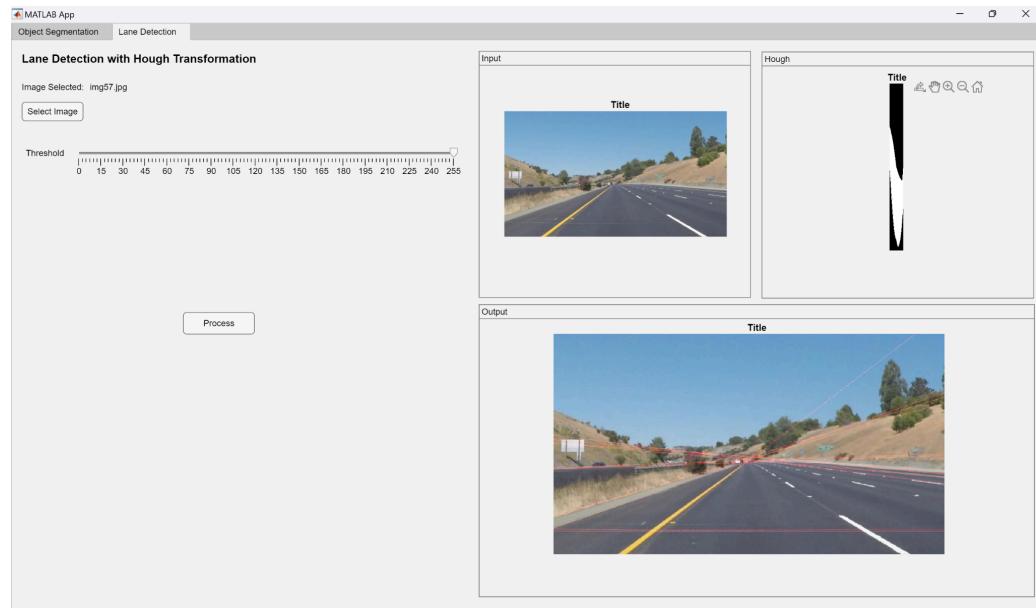




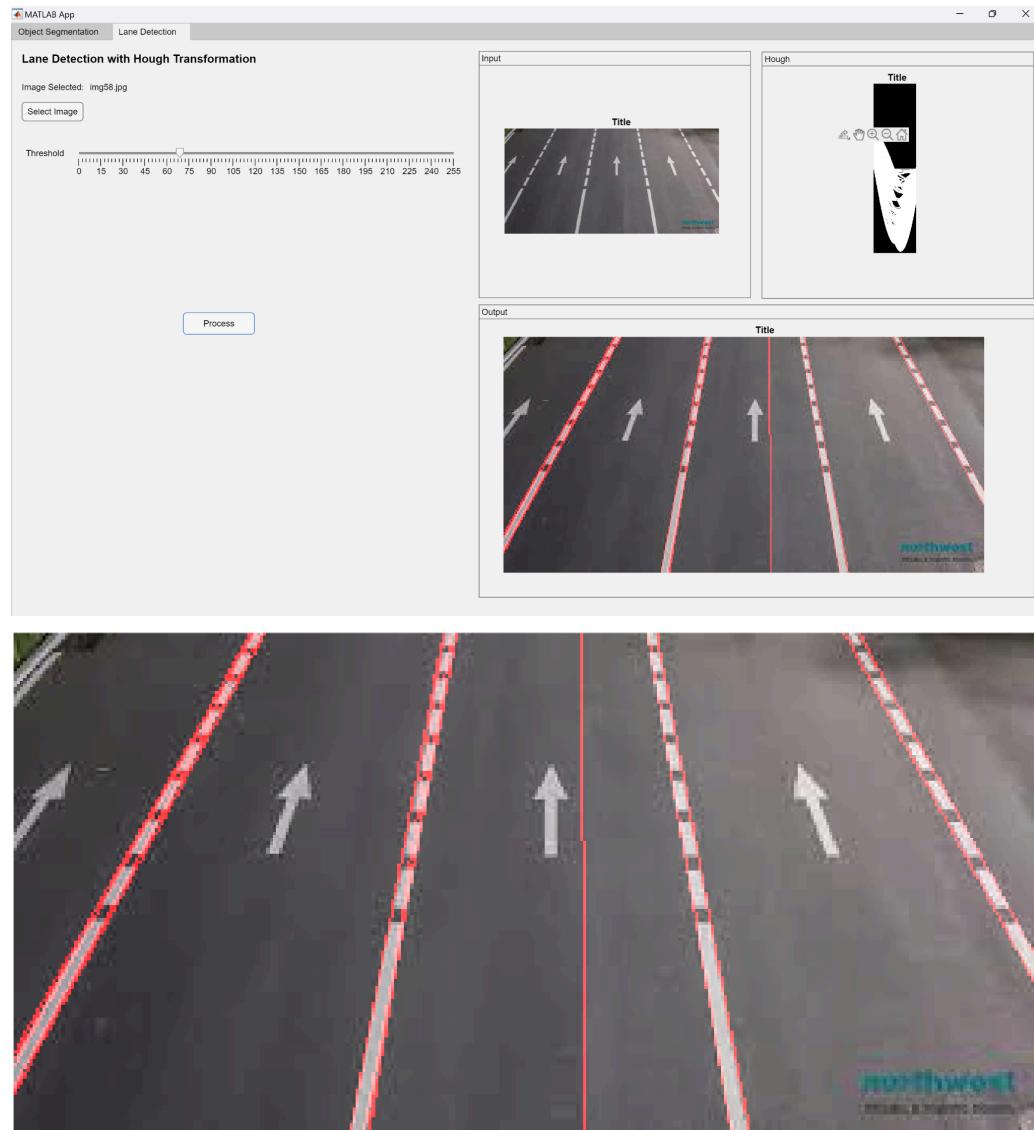
Gambar 3.2 Deteksi Garis Jalur dari Citra img39.jpg
dengan Bantuan Transformasi Hough ($T=240$)



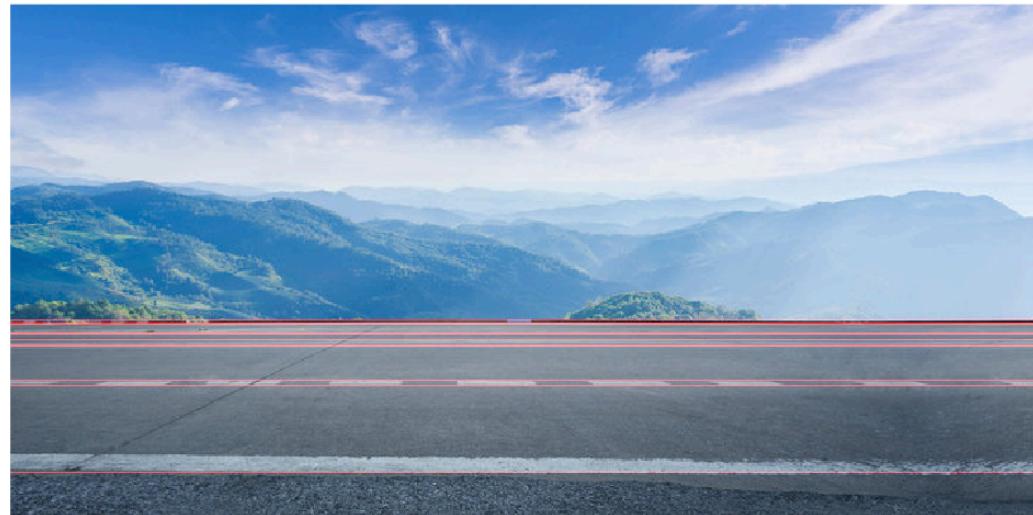
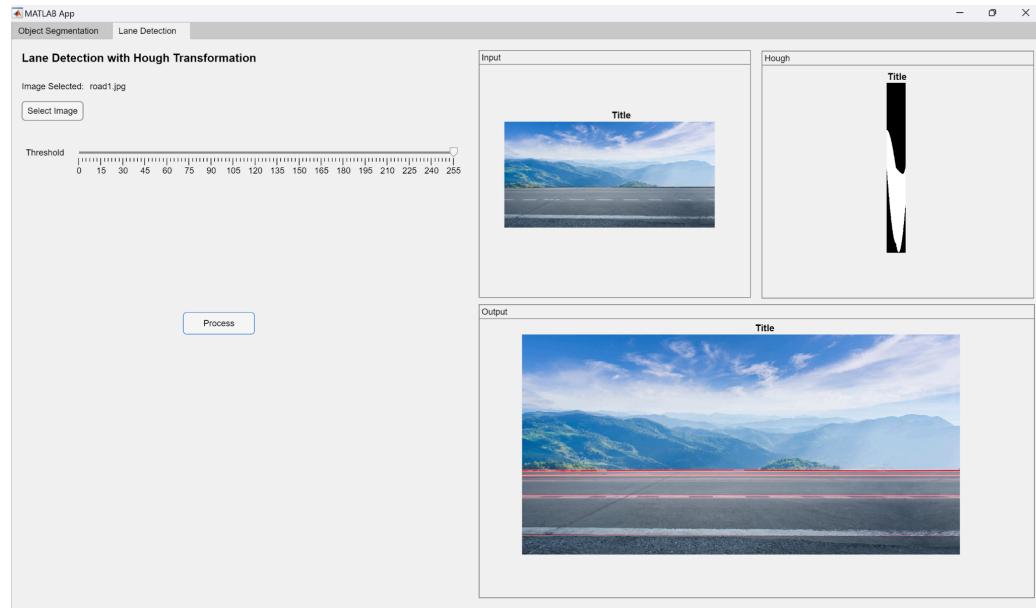
Gambar 3.3 Deteksi Garis Jalur dari Citra img40.jpg
dengan Bantuan Transformasi Hough ($T=255$)



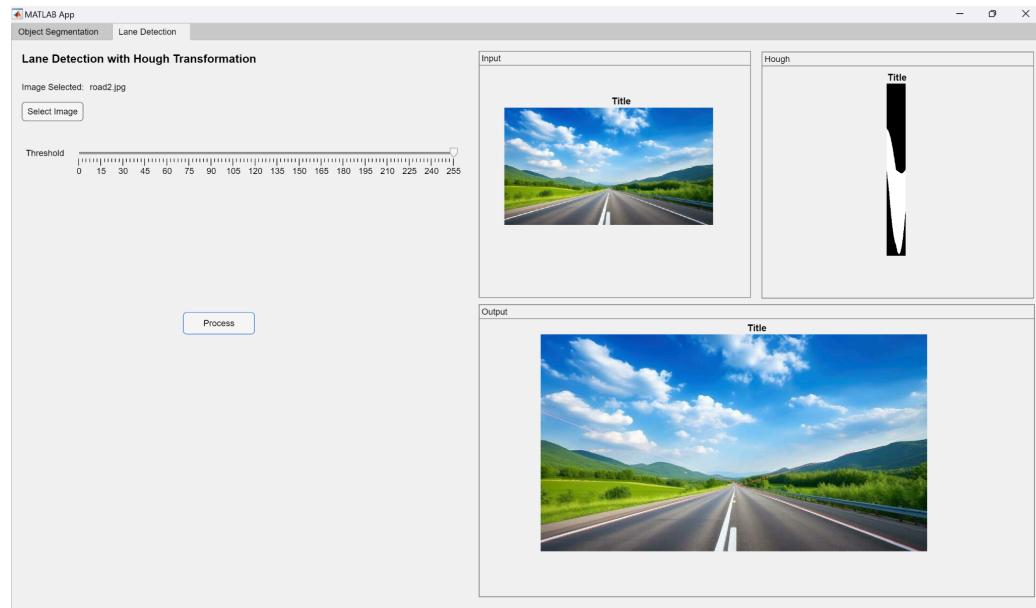
Gambar 3.4 Deteksi Garis Jalur dari Citra img57.jpg
dengan Bantuan Transformasi Hough ($T=255$)



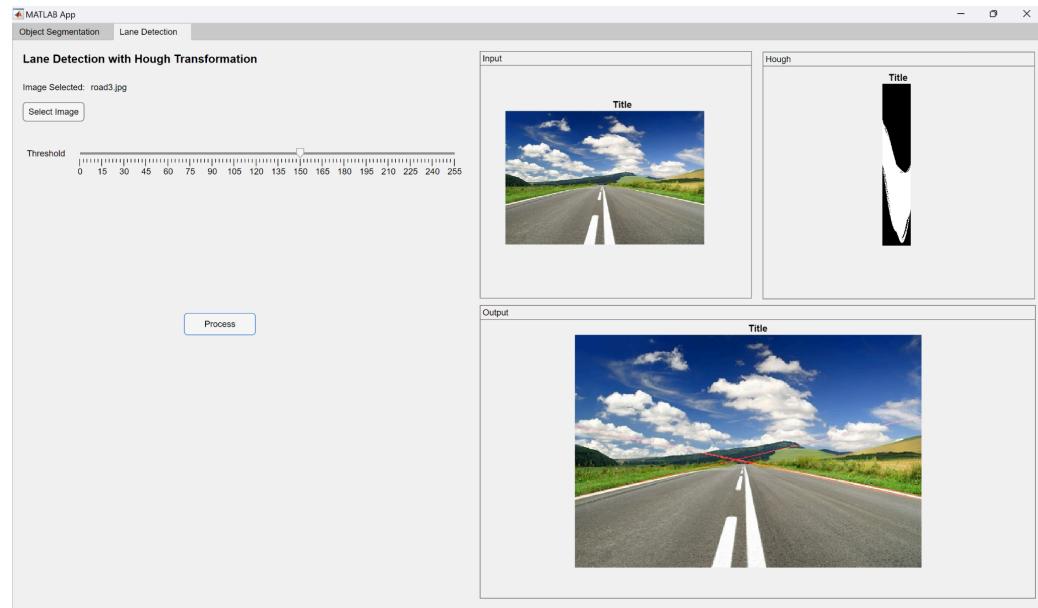
Gambar 3.5 Deteksi Garis Jalur dari Citra img58.jpg
dengan Bantuan Transformasi Hough ($T \approx 70$)



Gambar 3.6 Deteksi Garis Jalur dari Citra road1.jpg
dengan Bantuan Transformasi Hough ($T=255$)



Gambar 3.7 Deteksi Garis Jalur dari Citra road2.jpg
dengan Bantuan Transformasi Hough ($T=255$)



Gambar 3.8 Deteksi Garis Jalur dari Citra road3.jpg
dengan Bantuan Transformasi Hough ($T=150$)

Analisis:

Pada sebagian besar kasus uji, deteksi garis jalur jalan raya dengan bantuan transformasi Hough hanya dapat mendeteksi garis yang ada di kedua sisi jalan. Garis yang ada di tengah jalan tidak dianggap sebagai garis apabila T (*threshold / ambang*) bernilai tinggi. Fenomena ini mungkin bisa terjadi karena bentuknya yang lebih pendek atau lebih melengkung, sehingga tidak mendapatkan *vote* sebanyak garis di sisi jalan.

Sebaliknya, jika nilai ambang dibuat rendah, garis di tengah jalan terdeteksi, sekaligus garis-garis “imajiner” yang sebenarnya tidak ada. Beberapa benda seperti pagar dan jalur pejalan kaki terdeteksi sebagai garis “imajiner” karena bentuknya memang menyerupai garis. Hal ini menimbulkan dilema besar dalam memilih nilai untuk parameter T . Meskipun demikian, performa deteksi garis pada percobaan citra “img58” dan “road1” sudah sangat bagus karena semua batas jalan telah terdeteksi dengan sempurna.

IV. Alamat GitHub Program

https://github.com/kevinjohn01/IF4073_Tugas3.git